Validating Context-Driven Features of Mobile Applications Using Laboratory Testing

Chu Luo

Submitted in total fulfillment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
The University of Melbourne
Author ORCID: https://orcid.org/0000-0002-3814-1074

April, 2019

Abstract

Mobile devices such as smartphones and tablets now embed a variety of sensors that can provide context-driven services to increasingly large user bases. Driven by the richness of heterogeneous data and machine learning algorithms, context-driven mobile applications are becoming increasingly popular across a variety of domains, including location-based services, urban sensing and e-health. Although developers and researchers have proposed a significant number of mobile context development frameworks and testing tools to simplify the development and testing of context-driven mobile applications, it remains a challenge to efficiently and systematically validate the context-driven features of mobile applications. Real-world tests are often impractical, time-consuming and involve high cost.

In this thesis, we propose three approaches to alleviate these problems by enabling efficient and systematic validation of the context-driven features of mobile applications in low-cost laboratory settings. First, we present TestAWARE, a laboratory-based testing tool that facilitates efficient and systematic validation of context-driven mobile applications on Android. We demonstrate that TestAWARE can help testers validate functional properties and non-functional properties of mobile applications. Second, we propose a validation approach for the real-time sensing performance of context-driven mobile applications. We show that the performance properties of real-time sensing applications can be efficiently and systematically measured in the laboratory. Third, we present a validation approach that considers a variety of aspects of machine learning design in context-driven mobile applications. We demonstrate that testers can efficiently and systematically validate multiple machine learning design

choices in the laboratory. In addition, we discuss the strengths and weaknesses of laboratory testing techniques in general. We also provide some deliberations about the impacts of laboratory testing techniques on the software development process.

Our approaches are supported by theoretical investigation, empirical evaluation and peer-reviewed publications. Overall, this thesis has significantly enhanced existing methodologies to validate the context-driven features of mobile applications.

Declaration

This is to certify that

1. The thesis comprises only my original work towards the PhD.
2. Due acknowledgement has been made in the text to all other material used.
3. The thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Chu Luo

Preface

This thesis has been written at the School of Computing and Information Systems, The University of Melbourne. The major parts of this thesis are in Chapter 3, 4, and 5. Before I transferred my PhD study to The University of Melbourne, I completed all the work in Chapter 4 and some work in Chapter 3, 5 at University of Oulu, Finland. The studies presented in Chapter 3 and 5 were conducted at University of Oulu, Finland, under local ethical guidelines. The experiments presented in Chapter 4 did not involve any human/animal subject. Chapter 3, 4, and 5 are based on peer-reviewed publications and I declare that I am the primary author and have > 50% contributions in each of the following publications:

1. Chu Luo, Miikka Kuutila, Simon Klakegg, Denzil Ferreira, Huber Flores, Jorge Goncalves, Mika Mäntylä, and Vassilis Kostakos. TestAWARE: a laboratory-oriented testing tool for mobile context-aware applications. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):80, 2017.

2. Chu Luo, Angelos Fylakis, Juha Partala, Simon Klakegg, Jorge Goncalves, Kaitai Liang, Tapio Seppänen, and Vassilis Kostakos. A data hiding approach for sensitive smartphone data. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 557-568. ACM, 2016.

3. Chu Luo, Aku Visuri, Simon Klakegg, Niels van Berkel, Zhanna Sarsenbayeva, Antti Möttönen, Jorge Goncalves, Theodoros Anagnostopoulos, Denzil Ferreira, Huber Flores, Eduardo Velloso, and Vassilis Kostakos. Energy-efficient prediction of smartphone unlocking. *Personal and Ubiquitous Computing*, 23(1), pages 159-177, 2019.

Acknowledgements

My PhD study is both abstractly and effectively a peripatetic and marvellous journey across continents. All along I have been surrounded with care, support and enlightenment from my supervisors, colleagues, parents and my friends. Electricity powers my computer and you power me.

First and foremost, I would like to express my greatest gratitude to my supervisor, Prof. Vassilis Kostakos, who has kindly offered me this precious opportunity to undertake the PhD study in Finland and has kindly transferred me to Australia. When I arrived in Finland from China "over the hills and far away", and when I transferred to Australia from Finland "far from the forests and the snow", Vassilis never ceased to provide thorough support and inspiring guidance in my academic and personal life, as well as on the basketball court.

I would also like to express my appreciation to my supervisor, Dr. Jorge Goncalves, who helped me and guided my research in Finland and Australia. I am also very grateful to my supervisor, Dr. Eduardo Velloso, who assisted me and guided my research in Australia. I was studying with my three supervisors as if a car was navigated by three GPS satellites. In addition to my supervisors, I sincerely thank my advisory committee chair, Prof. Harald Søndergaard, who gave me kind care and insightful advice during my study in Australia.

My gratitude also goes to Prof. Frank Vetere, Zaher Joukhadar, Kat Franks, Julie Ireland, Rhonda Smithies, Dr. Nicole Barbee, Joshua Newn and Sarah Webber for helping me restart and complete my study at School of Computing and Information Systems, The University of Melbourne.

My research is supported by collaboration with multiple research units across continents. I am very grateful to Prof. Tapio Seppänen and Prof. Mika Mäntylä for leading their teams to assist my research. I would like to thank Dr. Angelos Fylakis, Dr. Juha Partala, Dr. Simon Klakegg, Dr. Kaitai Liang, Miikka Kuutila, Dr. Denzil Ferreira, Dr. Huber Flores, Dr. Aku Visuri, Niels van Berkel, Zhanna Sarsenbayeva, Antti Möttönen and Dr. Theodoros Anagnostopoulos

for contributing to my research. I would also like to thank Assoc. Prof. Tao Gu and Yu Zhang at RMIT University, Cynthia Yu and Dr. Hu Chen at Intel, Assoc. Prof. Yong Li at Tsinghua University, Dr. Fernando Koch at IBM for inviting me to join their research which strengthens my ability and knowledge in multiple domains of computer science. I would also like to pass my appreciation to Prof. Timo Ojala, Dr. Simo Hosio, Dr. Hannu Kukka, Dr. Marko Jurmu and Hannu Rautio for helping me with my study at University of Oulu, Finland. I would also like to thank my colleagues at The University of Melbourne: Dr. Tilman Dingler, Dr. Greg Wadley, Assoc. Prof. George Buchanan, Dr. Dana McKay, Danula Hettiachchi, Gabriele Marini, Senuri Wijenayake, Weiwei Jiang, Yitong Chen, Ruining Dong, Zewen Xu, Mengjia Chen, Chaofan Wang, Xiuge Chen and Yuan Wang for supporting or working together with me in multiple projects and academic events. My sincere thank also goes to my friends at The University of Melbourne: Xiaoxuan Tang and Bresson Li for helping me.

Besides, I gratefully acknowledge The University of Melbourne, Google and Academy of Finland as financial supporters for this thesis.

Finally, my great gratitude overflows from each atom of me to my parents for all the overwhelming love, encouragement and support beyond mountains and seas, over every nanosecond.

Contents

List of tables

List of figures

Chapter 1

Introduction

"Nobody actually creates perfect code the first time around, except me. But there's only one of me."

— Linus Torvalds

1.1 Motivation

Mobile devices, such as smartphones, tablets and smartwatches, are highly prevalent in developed countries [88, 89, 106, 133]. Also, as an increasing number of mobile devices are becoming inexpensive, the adoption of mobile devices is on the rise in developing countries [89, 101, 106, 133]. On these mobile devices, users can interact with various applications in mobile environments for many purposes, including social networking, business, online shopping, mobile gaming, etc. Meanwhile, hardware manufacturers have been embedding a variety of sensors (e.g., accelerometer, light sensor, barometer and heart rate sensor) into mobile devices.

Based on diverse sensors and other data sources (e.g., operating systems, hardware components, software applications and human user input) on mobile devices, a significant number of mobile applications can now process or even adapt to dynamic context information. Simple context-driven mobile applications, such as a sound recorder, only collect and process context information, without recognition or adaption. In contrast, advanced context-driven mobile applications can adjust their behaviours by analysing the dynamic context information so that they are context-aware [137].

With the prevalence of machine learning algorithms, mobile applications are able to provide many kinds of services with recognition of complex context [35,92]. For example, mobile applications on smartphones can recognise music or speech from audio streams [97], detect dangerous driving behaviours on cars [160], and monitoring Parkinson's disease [16, 91]. Likewise, context-aware mobile applications on smartwatches range from gesture recognition [157, 166], driver drowsiness detection [94], to cigarette smoking detection [140, 143].

Although the context-driven features of mobile applications can help users in various scenarios, validating these complicated applications is an extremely challenging task [56, 104, 152, 161]. Since these applications are designed for specific scenarios (e.g., particular time, locations and groups of users), it is often costly, in terms of time, money and resources, to perform tests in the real world [99, 152]. Hence, developers and researchers have proposed several approaches and tools that can simulate context to drive context-driven mobile applications for validation purposes in the laboratory. For instance, popular Integrated Development Environments (IDEs) such as Android Studio [59] and Xcode [8] contain several debugging functions for testers to manually simulate some types of context events (e.g., sensor readings and GPS locations) in development environments. Besides IDE-based testing tools, several specialised tools are proposed to validate context-driven mobile applications in laboratory settings, ranging from context replay tools (e.g., [21, 22, 95]), context record-and-replay tools (e.g., [56, 126]), to performance simulators (e.g., [30, 100, 110]).

However, previous work has several shortcomings:

1. Most approaches in previous work focus only on the examination of functional properties. Testers can hardly efficiently validate diverse functional and non-functional properties. With existing tools, it is time-consuming to conduct multiple rounds of regression testing (i.e., the examination of influences related to software changes [159]) when new functionality is integrated in a new version.

2. Previous work has little support for multiple ways of data acquisition. Note that some multi-dimensional context scenarios may need heterogeneous sensor data from multiple sources [99, 120].

3. The vast majority of existing approaches aim only to test implemented software applications. Few approaches help developers in validation (e.g.,

design phase testing) before implementation. Early detection of software flaws can reduce the complexity and cost for developers to repair [118, 142].

Consequently, in this thesis, we investigate new approaches to address these issues with regard to three research questions, as we discuss further.

1.1.1 Research questions

Driven by the shortcomings of previous work, the research questions of this thesis are the following:

- **RQ1. How can context-driven mobile applications be efficiently and systematically validated in the laboratory?**
- **RQ2. How can the real-time sensing performance of context-driven mobile applications be efficiently and systematically validated in the laboratory?**
- **RQ3. How can the machine learning design of context-driven mobile applications be efficiently and systematically validated in the laboratory?**

1.2 Contributions and author's role

Three publications are included as main contributions in this thesis. They are published in prestigious, international and peer-reviewed conferences/journals in the field of Ubiquitous Computing.

In Chapter 3, we present TestAWARE, a laboratory-based testing tool that facilitates efficient and systematic validation of context-driven mobile applications on Android. TestAWARE can help testers examine functional properties and multiple non-functional properties of mobile applications. To simulate context, TestAWARE supports the replay of sensor, event and audio data in a synchronised manner. TestAWARE also supports replay speed control that allows testers to specify a multiple of the original speed, so that TestAWARE can accelerate or slow down the simulation to quickly go through longitudinal datasets or to carefully examine complex executions. In addition, TestAWARE can import data from online databases or local sources. For the simulation of uncommon context events, testers can create arbitrary context data using data manipulation

functions. We demonstrate that TestAWARE can help testers perform efficient and systematic validation to detect and to locate flaws. The thesis author was responsible for the design and implementation of TestAWARE with co-authors, conducting experiments, and data analysis.

In Chapter 4, we focus on the validation of real-time sensing performance of context-driven mobile applications. We show that the performance properties of real-time sensing applications can be efficiently and systematically measured in the laboratory at the design phase with only software design and limited implementation efforts. We present an approach that considers multiple aspects of performance validation, including quantifying the amount of error caused by modification in sensor signals, measuring the processing speed and CPU utilisation of executions, measuring the performance change caused by different sensing frequencies, sensor types and device models. With a smartphone sensor-based real-time data hiding method as an exemplar case, we highlight the effectiveness of our approach. The thesis author was responsible for the design and implementation of the data hiding method with co-authors, designing the validation approach to conduct tests, and data analysis.

In Chapter 5, we focus on the validation of machine learning design of context-driven mobile applications. We present an approach that takes into account a variety of aspects of machine learning design. We demonstrate that, in some scenarios, applications can make use of either of the two machine learning tasks: classification and regression. Based on datasets obtained in the real world, our approach helps testers conduct design phase testing in the laboratory to measure the predictive performance of different algorithms for both classification and regression, to quantify the importance of each feature, to compare models built by software-generated features and all features, to compare personalised models and cold start models in classification, and to quantify energy consumption. We select as an exemplar case a smartphone-based system relying on machine learning algorithms and contextual data to predict the next unlock event triggered by users. With the selected exemplar case, we demonstrate the effectiveness of our approach. The thesis author was responsible for the design and implementation of data collection tool used in the exemplar case, designing the validation approach to conduct tests, and data analysis.

1.3 Thesis outline

We organise the remainder of the thesis as following.

In Chapter 2, we provide a background of related work and concepts. We summarise the characteristics of modern mobile devices in terms of their essential components, sensors, operating systems and mobile applications. Then, we introduce mobile applications with context-driven features by defining context in mobile computing and the features driven by context. Finally, we look at existing validation approaches including real-world tests and laboratory testing techniques.

In Chapter 3, we detail the design and implementation of TestAWARE, a laboratory-based testing tool that facilitates efficient and systematic validation of context-driven mobile applications. This chapter is to address RQ1.

In Chapter 4, we present an approach that covers multiple aspects of real-time sensing performance validation of context-driven mobile applications in the laboratory. This chapter is to address RQ2.

In Chapter 5, we present an approach to efficiently and systematically validate machine learning design of context-driven mobile applications in the laboratory. This chapter is to address RQ3.

In Chapter 6, we first revisit the research questions. Then we discuss the strengths and weaknesses of laboratory testing techniques. We also provide some deliberations about the impacts of laboratory testing techniques on the software development process. Finally, we summarise the limitations of this thesis and give some future directions with suggestions.

Chapter 7 concludes this thesis.

Chapter 2

Background

In this chapter, we provide a background of our research by summarising previous work in the literature. First, we briefly discuss the characteristics of modern mobile devices. We then give an overview of mobile applications with context-driven features. Regarding these applications, we review existing tools, techniques and strategies for the validation of context-driven features in both functional and non-functional aspects. Finally, by pointing out the limitations in the state of arts, we show the room for improvement that leads our research.

2.1 Characteristics of modern mobile devices

Mobile devices, such as smartphones, tablets and smartwatches, are adopted by the majority of people in developed countries, thanks to their portability, ubiquitous network access, multifunction and affordable price [10, 89]. In this section, we discuss the characteristics of modern mobile devices from the hardware and software perspective.

2.1.1 Essential components

Deriving from regular computers, mobile devices consist of essential modules such as Central Processing Unit (CPU), memory and hard disks. On recent smartphones and tablets, processors contain both CPU and graphics processing unit (GPU). For example, Qualcomm Snapdragon 845 processors [127] contain 8 Qualcomm Kryo 385 CPUs and an Adreno 630 GPU. Mobile devices with these processors can provide high performance and power efficiency for graphical

applications, including mobile games and VR (Virtual Reality) [117]. Mobile devices may also contain audio processors for audio playback.

Most mobile devices contain adaptors of various wireless networks (e.g., Wi-Fi [4] and Bluetooth [141]) for Internet access and communications with other devices. Especially, smartphones can access cellular networks for data roaming and phone calls via their modems (e.g., LTE [32]).

Although containing several physical buttons, mobile devices usually receive user input from their capacitive touch screens. Beyond displaying content, such touch screens allow users to input information by tapping or gestures [144]. Some types of touch screens support multi-touch gestures by several fingers [86].

Unlike regular computers with alternating current power supply, mobile devices are powered by battery. However, due to the small size of mobile devices and their battery, the battery life is generally short in daily usage. Most smartphone or smartwatch users have to recharge their devices every day [45, 109].

Most mobile devices use USB (Universal Serial Bus) [53] for recharging. USB can also be used for data exchange. Some mobile devices can be recharged using wireless charging technology which generates an electromagnetic field [71].

2.1.2 Sensors

Besides essential modules on every computer, modern mobile devices comprise a wide variety of built-in sensors. Android classifies sensors into three categories [64]: motion sensors, environmental sensors, and position sensors. Considering other sensors beyond the three categories, we summarise sensors of mobile devices in Table 2.1.

Besides hardware sensors, previous studies proposed a broad sense of "sensors" including software and human [89, 132, 151]. For example, the context instrumentation middleware AWARE [47] defines several software-based sensors such as application usage and installation, and a human-based sensor that collects user input data from questionnaires. Also, information generated by some hardware modules can be considered as sensor data. For instance, although a CPU is not a sensor, AWARE considers it a software-based sensor by collecting values of CPU system, user and idle workload.

	Sensor	Description of Measurement
Motion	Accelerometer	Three-dimensional acceleration
	Gravity	Three-dimensional gravitational acceleration
	Gyroscope	The rate of device rotation in three dimensions
	Rotation	Three-dimensional rotation vector of the device
Environmental	Temperature	Ambient air temperature
	Barometer	Atmospheric pressure
	Humidity	Relative ambient humidity
	Light	The intensity of light
	RGB sensor	The intensity of red, green and blue light
Position	Orientation	The orientation angle of the device in three dimensions
	Magnetometer	Three-dimensional ambient geomagnetic field
	Proximity	Proximity of an object relative to the device screen
	Location	Satellite-based or network-based location information
Other	Camera	Visual images and videos
	Microphone	Ambient sound
	Heart rate	Heart rate signals
	Battery	Battery status and events
	Bluetooth	Bluetooth status and scanned nearby devices
	Wi-Fi	Wi-Fi status and scanned nearby access points
	NFC	Near-field communications with nearby devices or tags
	Modem	Cellular network status

Table 2.1: Sensors on mobile devices.

2.1.3 Operating systems

Currently, there exist two popular operating systems (OS) for mobile applications: Android [63] and iOS [7]. Their applications are written in high-level object-oriented programming languages that are resilient to bugs in code.

Android is a Linux-based mobile operating system developed by Google. It supports phones, tablets, smartwatches and many other devices. Android applications are written in Java or Kotlin language. Application code for earlier Android versions was compiled to bytecode for running on Dalvik virtual machine. This mechanism often causes slight delay in executions as bytecode cannot be directly executed on device. From Android 5.0, Android Runtime (ART) replaced Dalvik to achieve higher performance by translating bytecode into native code [60]. As an open source platform, Android has a significant number of variants that slightly differ across manufacturers. Together with difference in

hardware modules of these manufacturers, Android strongly challenges developers on the fragmentation problem [67].

iOS is a mobile operating system developed by Apple, supporting various mobile devices including: smartphones, tablets and music players. Unlike the stable language selection in developing Android applications, the early language choice for iOS applications was Objective-C which expands C into an object-oriented language. However, Objective-C does not support advanced features such as automatic memory management and functional programming. Developers often have to manually implement many types of low-level actions. To overcome multifarious drawbacks of Objective-C, Swift succeeded Objective-C in 2014, while applications written in Objective-C are still compatible to operate on new iOS platforms. As only a small number of models of devices run iOS, developers need not face the fragmentation problem.

Other mobile platforms have also gained a little market share. For example, Windows Mobile/Phone [107] is the series of mobile operating systems developed by Microsoft. They are driving several types of smartphones (e.g., Nokia phones) and tablets. Tizen [54] is a Linux-based mobile operating system developed by the Linux Foundation, Intel and Samsung. Tizen mainly operates on smart TVs and wearables. Unlike Android and iOS, these unpopular mobile platforms receive insignificant attention from mobile developers and users.

2.1.4 Mobile applications

Mobile devices usually have pre-installed applications (e.g., settings, browser, music player, image viewer and camera app) for several basic functions. Among these pre-installed applications, an application market app (e.g., Google Play for Android, Apple Store for iOS) allows users to find, install, update and delete third-party applications. Some third-party applications are free to use, while the rest require users to make payments via the application market app.

The categories of mobile applications are highly diverse: social networking (e.g., Facebook [43]), games (e.g., Angry Birds [41]), education, business, shopping, etc. Due to the mobility of users carrying mobile devices, most mobile applications have to operate in varying environments. For instance, a big challenge for mobile applications is frequent disconnection of networks [52]. The signals of wireless networks may be temporarily blocked when users are trav-

elling in cities. The changing context may influence some behaviours of mobile applications, as we discuss in the following section.

2.2 Mobile applications with context-driven features

In this section, we first review how context is defined in the literature of mobile computing. Then, we discuss the context-driven features in mobile applications.

2.2.1 Context in mobile computing

Context is a popular concept in many domains. In psychology, context may refer to a personal or public environment where one or a group of individuals live or interact with each other [87]. In the field of mobile computing, researchers have proposed various definitions of context.

In an early work by Schilit and Theimer [138], context represents locations, nearby human and object identities, as well as temporal changes of objects. They further introduced the term "context-aware computing" to describe applications designed to discover and adapt to their contexts. Based on this definition of context, researchers developed amendments by integrating additional factors such as time of day [134] and user intent [34]. However, Dey et al. [36] pointed out that these definitions follow a scenario-specific basis and deliver limited reusability for different types of context-aware applications.

Consequently, in this thesis, we adopt a definition by Abowd et al. [1] which has more universality across diverse scenarios: "any information that can be used to characterise the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves". This definition is extensively used in the field of mobile computing.

Moreover, in the practice of designing specific applications, designers often tend to consider context a narrow concept. The meanings of context are essentially open-ended between objects or activities [37] . Context can be defined dynamically in particular settings, because the relevance of a context feature may vary during the interaction between users and applications. Considering the dynamic constraints of obtaining specific context features, some advanced

machine learning algorithms, such as Feature-Budgeted Random Forest [114], can improve the stability of performance by selecting a suitable set of features just before a prediction. Also, beyond objective notions of context, context includes intersubjective aspects such as experience of individuals [28]. Although designers of context-aware applications often emphasise objective context, integrating intersubjective context with objective context may improve the relevance of computational results for various user activities [28]. Since tracking intersubjective context relies on analysis on objective context, designers can achieve such an integration without using additional sensors or data sources [28].

Hence, despite the open-ended meanings of context, the data sources for tracking context information on mobile devices are hardware sensory readings, software data and human input. The definition by Abowd et al. [1] provides the generality to describe various mobile applications.

2.2.2 Context-driven features in mobile applications

Except for stand-alone applications such as calculator, mobile applications generally contain context-driven features.

Context-driven features are not necessarily context-aware features [125]. For instance, smartphones usually have sound recording applications that record ambient sound as audio files or voice messages. Although operating with contextual data, such applications do not recognise and adapt to context.

Context-aware features involve context detection and subsequent adaption. A typical example is that email applications on mobile devices automatically synchronise to servers when mobile network connection (cellular network or Wi-Fi) is available [2]. Moreover, mobile applications may use contextual data and machine learning algorithms to recognise high-level context for adaptive decision making. For example, CarSafe [160] is a smartphone-based application aiming to ensure safe driving. From the vehicle windscreen mounting the smartphone, CarSafe tracks driver behaviours using the front camera and monitors road conditions using the rear camera. Besides computer vision techniques, it uses GPS, accelerometer and gyroscope to track the motions of the vehicle. CarSafe notifies the driver by generating alerts if it detects risky driving actions. Other context-aware features of mobile applications include gesture recognition [166],

location-based advertisements [129], augmented reality [155], natural-language voice interaction [18], etc.

2.3 Existing validation approaches

2.3.1 Real-world tests

To validate context-driven features of mobile applications, testers can choose to perform real-world, also known as in-the-wild, tests in the intended context scenarios. For example, CarSafe [160] was evaluated in the real-world conditions where drivers were asked to drive cars on the road.

To conduct real-world tests on an application, developers first need to select a field setting which serves as the intended context scenario [84, 152]. Also, if the application is designed to interact with human users, developers have to recruit a group of participants to use the application. For different purposes of applications, developers have to identify the characteristics of the target user group to recruit suitable participants. Then, developers should give tasks of application interactions to the recruited participants. When participants are performing tasks, developers have to monitor the performance of the application. For example, developers can record runtime states of the application using a background service on the mobile device.

In addition, some mobile applications, such as navigation systems [29] , are designed to operate without explicit user interactions. In these cases, developers have to give commands directly to applications on mobile devices in the intended context scenarios. For example, in real-world environments, developers can test navigation systems by comparing the ground-truth and reported locations of deployed mobile devices.

Real-world tests offer high realism and generalisability in natural settings without relying on laboratory equipments or technological resources [39, 84, 152]. Thus, due to low technical barriers, real-world tests can be performed by people who have limited technical skills. Regarding the stages of development, real-world tests can be applied for understanding the requirements or for evaluating applications [84]. Particularly, real-world tests are often used with lightweight prototypes to support application design in early stages [39, 84]. In contrast,

for the evaluation stage, the majority of researchers and developers applied laboratory testing [84].

Despite the high fidelity of contextual data collected in real-world tests, there exist several issues, as illustrated in multiple studies [99, 152]:

1. **Participant recruitment.** Different user characteristics, such as age, gender, nationality and impairments, may affect the behaviours of tested applications. Certain flaws may only happen to a specific group of users. However, recruiting a sufficient number of representative participants can cause high cost which is not affordable for developers.
2. **Context availability.** Some context events (e.g., dangerous driving behaviours [160], low temperature with constant wind speed [58]) are very rare and difficult to artificially produce in the real world.
3. **Time constraints.** Some real-world tests require a significantly long period of time. For example, validating diabetes monitoring applications needs longitudinal datasets over 6 months [27].
4. **Resource constraints.** Some resources (e.g., battery) on mobile devices are too limited to support extensive testing in the wild.

2.3.2 Laboratory testing

Instead of conducting tests in the real world, developers and researchers have proposed several approaches and tools to perform tests in laboratory settings. In laboratory testing, contextual data is simulated and delivered to the context processing modules modules of mobile applications, so that context-driven features can be triggered and validated. These laboratory testing methods can be classified into the following three main categories.

IDE-Based Context Simulation Integrated Development Environments (IDEs) of major mobile platforms provide basic support for context simulation in development environments. For example, developers can manipulate several system events (e.g., phone calls) and sensor readings (e.g., acceleration) to an emulator via Android Studio [59] which is the IDE for Android applications. Figure 2.1 shows the interface to simulate several types of contextual data for an emulator on Android Studio. Also, similar features are integrated in Xcode [8] which is the

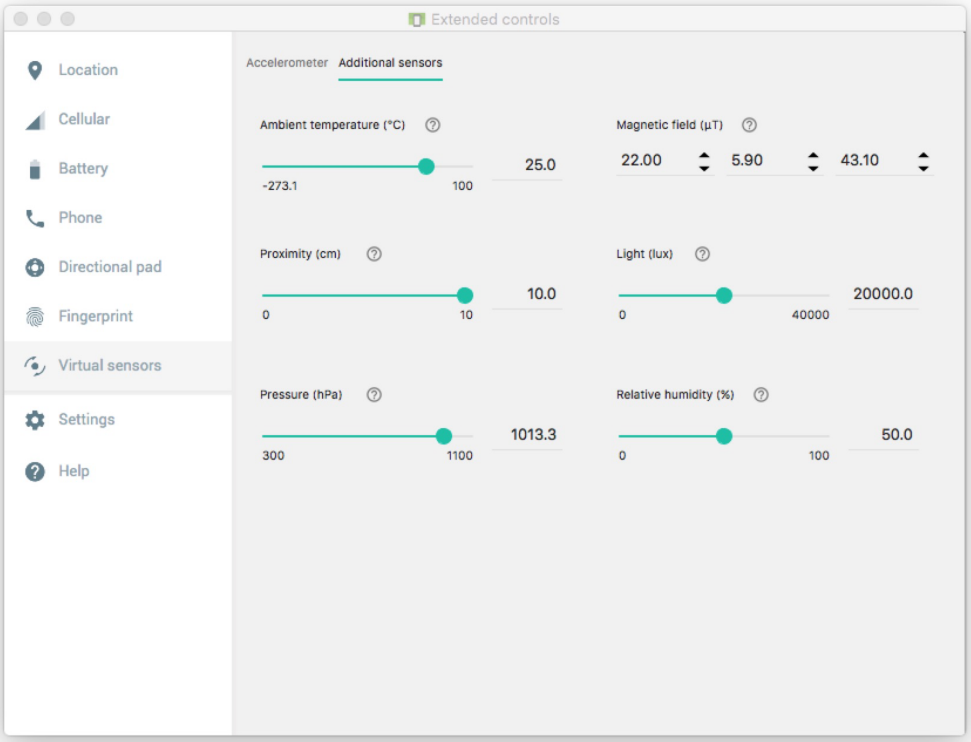


Figure 2.1: Context simulation interface for an emulator on Android Studio.

IDE for iOS applications. For instance, Figure 2.2 shows the interface to simulate locations for an emulator on Xcode.

However, these context simulation interfaces require manual input from developers and cannot support some test types such as stress testing (e.g., processing data from sensors running at high frequency) and longitudinal tests (e.g., simulating a dataset involving data collected over many weeks). Also, manual input implies significant efforts to repeat tests in regression testing.

Hence, some IDEs provide additional support to automate context simulation. For instance, the Android platform contains a command-line tool called Monkey [61] which allows testers to generate a large number of pseudo-random user interface (UI) events or several system events on an emulator or physical device. With a random number seed, Monkey can reproduce an event consequence to repeat a test. In addition, Android provides another tool called monkeyrunner [62] which contains a Python API (application programming interface) for testers to write scripts to automate complex testing processes, such as installing packages, generating events and capturing screenshots.

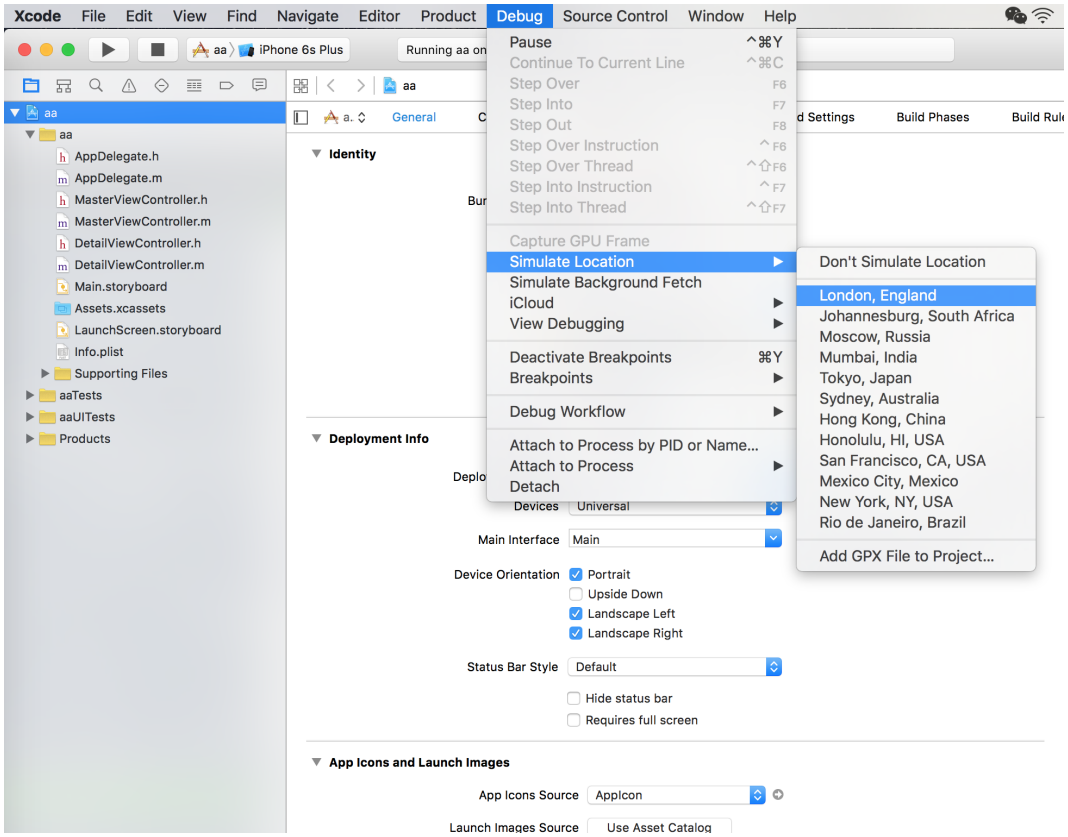


Figure 2.2: Location simulation interface for an emulator on Xcode.

Third-Party Context Simulation Outside development environments, there also exist several third-party context simulation approaches for validating context-driven features of mobile applications. Amalfitano et al. [6] proposed and evaluated the concept of simulating both context and UI events to test mobile applications. They implemented this concept with several test case generation techniques to simulate events by executing JUnit scripts [80] on a device or emulator. Through experiments on real-world applications, they found that simulating context events together with UI events is an effective way to increase code and method coverage, compared to simulating UI events only. She et al. [139] integrated network states and UI events into context simulation on a testing platform called Hermes. Hermes reads test data by parsing XML schemas [154] which can be written manually or generated using a software program. PUMA [69] also reads test data from scripts. It relies on a customised scripting language called PUMAScript which can represent UI and system events.

Considering that specifying test cases of context data in scripts is inefficient, several approaches, such as MobileTest [21] and Caiipa [95], allow testers to import datasets from a database. Also, beyond UI and system events, MobileTest and Caiipa can simulate context changes using sensor readings and network states. Similarly, KnowMe [22] can simulate context for PC-based Java programs using datasets collected by the middleware AWARE [47].

Moreover, a significant number of approaches rely on self-generated context data, without importing external datasets. In [103], a generator is used to randomly produce context data, including system events, sensor readings and network states. Testers can add assertions into code blocks to examine application states after context changes. Furthermore, considering the large number of possible combinations of contextual data values, Liu et al. [96] presented an adaptive random testing technique which minimises similarity among test cases in random generation to increase the test coverage. VanarSena [131] also relies on randomly generated context events. During context simulation, VanarSena takes into account the frequency of inducing unexpected events. For example, to test an Internet-based application, VanarSena generates error status code (e.g., 400 Bad Request and 500 Internal Server Error) with a low probability, since events with too many errors cannot efficiently explore the intended working states of the targeted application.

Besides random generation, several advanced techniques can also be used to automatically generate context data:

1. **Sensing offloading.** Sensing offloading is usually used to support battery-limited devices to acquire sensor data from nearby devices. For instance, many sensing offloading systems and studies (e.g., [49–51, 128]) aim at context-aware applications on personal mobile devices such as smartphones, due to their relatively low battery capacity. Unlike systems using offloaded sensor data to infer context, RainDrops [165] extends the sensing offloading technique to collect sensor data for testing mobile applications on a cloud of physical devices and emulators. During tests, RainDrops runs and examines the targeted application on emulators. When the application invokes an API call to collect sensor data, RainDrops offloads the sensing task to physical devices. To support reliable continuous sensing offloading, RainDrops creates a buffer of heterogeneous sensor data collected from physical devices in the real world, so that the sensor data can be prepared before tests.

2. **Static analysis.** Static analysis is a widely used technique of software testing which examines the syntax and semantics of source code without executions on hardware [11, 15]. To test mobile applications, several approaches adopt static analysis to generate relevant contextual data to exercise context-driven features. In [145], the test case generation process considers the permission list of a targeted application. This permission list contains the requested use of hardware modules (e.g., Bluetooth) or information resources (e.g, location) which are sensitive to user privacy and system security. Based on this permission list, context events (e.g., the user enables a hardware module) that are relevant to the behaviours of the targeted application can be identified. By combining the possible states of each context type using Cartesian product, the universal set of all test cases can be obtained. For different testing objectives (e.g., validating a specific behaviour), a selected subset of all test cases may be used to avoid unnecessary validations on irrelevant input. Besides the permission list, in [153], the static code analysis also extracts the event handlers and callback functions of UI and system events from the application bytecode. Hence, the generated test cases can contain both permission-related context events and permission-independent events (e.g., UI and insensitive system events). Together with static analysis, dynamic analysis is also used in ATT [105]. When applications launch context-related system services at runtime, ATT can automatically update its generation process on the fly to produce contextual data for the newly created services.

3. **Fuzzing.** Fuzzing, also known as fuzz testing, sends random input, including valid and invalid values, to targeted applications [17, 149]. The input generators of fuzz testing are called fuzzers. Chizpurfle [72] is a fuzz testing tool for vendor-specific services on Android. To generate input for targeted services, its fuzzer covers four types of service parameters: primitive types (e.g., float and integer), strings, arrays/lists, and objects. After the targeted service executes a test case, Chizpurfle tracks new code blocks in the execution. In the next execution, Chizpurfle generates a test case by mutating the previous test case that activates the execution of new code blocks. Beyond code coverage, Chizpurfle also tracks log messages (e.g., failed assertions and fatal exceptions) at runtime to detect errors. Similarly, Qui-Gon Jinn (QGJ) [158] is a fuzz testing tool for wearable applications. Besides systems events and messages, QGJ also generates random UI events. After sending generated input, QGJ monitors the application and operating system in terms of errors, crashes, reboots and hangs. In experiments with real-world applications, some malformed unprivileged messages caused reboots of operating systems on devices.

In addition to context simulation for functional testing, several approaches simulate contextual data for non-functional evaluation and optimisation. For instance, Kobe [30] is a performance simulator that aims to balance the tradeoff among energy consumption, latency and accuracy of mobile sensing systems. Instead of running actual applications, Kobe estimates the energy consumption and latency by executing the classification algorithms in cloud-based emulation and regression models of performance. PADA [110] is a power estimation tool for mobile sensing applications. For power analysis, it allows users to upload an implemented application with a context scenario (e.g., the user is walking for 5 minutes in outdoor space). Then, it executes the application and emulates the scenario with contextual data collected in previous real-world experiments. Finally, it reports the power consumption over time and patterns of each hardware module. FOREPOST [100] is a performance bottleneck analyser for Java applications. It executes the targeted application with random input data to identify low-performance execution paths and potentially improvable methods.

Record-and-Replay Tools For certain context scenarios, it may be inconvenient for testers to find historical data or to generate suitable data to perform simulation in laboratory settings. Hence, testers sometimes have to conduct a real-world

experiment for data collection. To simplify the process of data collection and subsequent simulation, several tools provide record-and-replay functions, so that testers can record context data in a real world experiment and reuse the recorded data to repeat tests.

For example, RERAN [56] is an Android-based record-and-replay tool that can capture touchscreen input, sensor readings and system events during application usage. By replaying the recorded trace, RERAN can reliably reproduce bugs that are manually produced. The record-and-replay functions of RERAN require rooting only, which gives users privileged control (similar to the superuser privilege in Linux) of Android OS. However, RERAN cannot record some types of sensor data from Android system services, such as GPS-based or network-based location information. Hence, MobiPlay [126] overcomes this weakness by connecting the testing mobile device to a server that runs a modified Android OS on a high-speed network. The targeted application is installed on the server, rather than the mobile device. The mobile device runs the MobiPlay client to show the UI of the targeted application and to receive user input. The user input is transmitted to the targeted application on the server via the high-speed network. Since the server runs a modified Android OS, sensor readings from system services and other events can be captured. Likewise, Paranoid [124] uses the device-server architecture that detects security attacks on the server using context data uploaded by a smartphone.

Unlike approaches recording data from only one mobile device, MOTIF [57] adopts a crowdsensing mechanism that collects data from a large group of devices in the wild. With a client installed on mobile devices, MOTIF records application information (e.g., method names), exceptions and contextual data. The client uploads recorded data to a cloud server where context patterns are extracted from aggregate data to reproduce crashes.

Summary Table 2.2 compares the characteristics of existing laboratory testing approaches for validating context-driven features of mobile applications. Most approaches in prior work focus only on the examination of functional properties. To test various performance properties, testers have to repeat tests with multiple specialised tools. However, repeating tests implies high cost of time and resources. Note that developers of the modern software industry tend to push daily and weekly commits in software development [42]. Repeating tests for frequent

Approach	Data Acquisition	Data Types	Properties Checked	Operating Environment	OS/App Modified
Monkey [61]	random generator	UI event, system event	functionality	device/emulator	no
monkeyrunner [62]	Python script	UI event, system event	functionality	device/emulator	no
[6]	JUnit script	UI event, system event	functionality coverage	device/emulator	no
Hermes [139]	XML	UI event, network	functionality	device/emulator	no
PUMA [69]	PUMAScript	UI event, system event	functionality performance	device/emulator	app
MobileTest [21]	database	UI event, system event sensor, network	functionality	device	no
Caiipa [95]	database	UI event, system event sensor, network	functionality performance	device/emulator	OS
KnowMe [22]	AWARE [47]	system event sensor, network	functionality	PC	no
[103]	random generator	system event sensor, network	functionality coverage	device/emulator	app
[96]	random generator	UI event, system event sensor	functionality	PC+emulator	no
VanarSena [131]	random generator	UI event sensor, network	functionality	emulator	app
RainDrops [165]	sensing offloading	system event sensor, network	functionality performance	device+emulator	OS
[145]	static analysis	system event, network	functionality	device	no
[153]	static analysis	UI event, system event sensor, network	functionality coverage	device	no
ATT [105]	static analysis	UI event, system event sensor	functionality coverage	device+emulator	app
Chizpurfle [72]	fuzzing	system event	functionality coverage	device	app
QGJ [158]	fuzzing	UI event, system event	functionality	device	no
Kobe [30]	database	sensor	performance	cloud	no
PADA [110]	database	sensor	performance	cloud	no
FOREPOST [100]	random generator	system event, sensor	performance	PC	no
RERAN [56]	recording	UI event, system event sensor	functionality	device	no
MobiPlay [126]	recording	UI event, system event sensor	functionality	device+server	OS
Paranoid [124]	recording	system event sensor, network	security	device+server	OS
MoTiF [57]	recording	UI event, system event sensor, network	functionality	device+cloud	no

Table 2.2: Characteristics of existing laboratory testing approaches for validating context-driven features of mobile applications.

commits strains testing resources. Although a few functional-testing approaches provide some support for non-functional testing, the covered non-functional properties are limited (e.g., Caiipa [95] has only energy consumption estimation).

Besides, all approaches in prior work allow only one way of data acquisition. This poses a critical challenge for testers to simulate multi-dimensional context with heterogeneous sensor data, since mobile applications are becoming increasingly complex. Also, sometimes testers have to integrate data from different sources (e.g., manipulated data and recorded data) to simulate the intended context in testing [99].

In addition, regarding testing phases, the vast majority of existing approaches aim only to test implemented software applications. Few approaches support design phase testing (e.g., Kobe [30] can examine the performance of machine learning classifiers at the design stage). In practice, design phase testing plays a crucial role in successful testing [118]. If flaws are found early in design documents, it is much easier and cheaper for developers to fix them [142].

Consequently, in this thesis we present approaches to address these identified issues in the literature. Our approaches aim to facilitate efficient and systematic tests on context-driven features of mobile applications in laboratory settings. Also, our approaches attempt to enable low-cost tests that examine performance properties of context-driven mobile applications at the design stage.

Chapter 3

A laboratory-oriented testing tool for context-driven mobile applications

3.1 Aims and hypotheses

This chapter presents TestAWARE, a testing tool that facilitates efficient and systematic validation of context-driven mobile applications in the laboratory. TestAWARE aims to overcome the drawbacks of prior work in the following aspects:

1. **Data type.** TestAWARE supports the replay of all sensor and event data types provided by AWARE (see AWARE sensor list [9]). Also, TestAWARE can replay audio streams in synchronisation with sensor and event data. For all data types, TestAWARE allows testers to set a replay speed faster or slower than the real clock.

2. **Data source.** To enable data reuse, TestAWARE allows testers to import data from online or local sources. TestAWARE also provides data manipulation for testers to create arbitrary data that is difficult to obtain from the real world or databases.

3. **Black-box testing.** For applications receiving data in the AWARE format, testers can use TestAWARE UI to launch functional testing in a black-box manner without writing scripts.

4. **White-box testing.** By recording application output and assertion results, TestAWARE allows testers to check low-level details in a white-box manner.

5. **Non-functional testing.** For all mobile applications, TestAWARE can measure processing speed of executions. For mobile sensing applications, TestAWARE can estimate power consumption of sensors on a specific device model. For context-aware mobile applications, TestAWARE can measure machine learning performance over runtime.

We hypothesised that TestAWARE can efficiently and systematically validate context-driven mobile applications in the laboratory. The design and evaluation of TestAWARE are detailed in the attached publication in Section 3.2.

3.2 Publication

Chu Luo, Miikka Kuutila, Simon Klakegg, Denzil Ferreira, Huber Flores, Jorge Goncalves, Mika Mäntylä, and Vassilis Kostakos. TestAWARE: a laboratory-oriented testing tool for mobile context-aware applications. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):80, 2017. `https://doi.org/10.1145/3130945`.

TestAWARE: A Laboratory-Oriented Testing Tool for Mobile Context-Aware Applications

CHU LUO, The University of Melbourne
MIIKKA KUUTILA, SIMON KLAKEGG and DENZIL FERREIRA, University of Oulu
HUBER FLORES, University of Helsinki
JORGE GONCALVES, The University of Melbourne
MIKA MÄNTYLÄ, University of Oulu
VASSILIS KOSTAKOS, The University of Melbourne

Although mobile context instrumentation frameworks have simplified the development of mobile context-aware applications, it remains challenging to test such applications. In this paper, we present TestAWARE that enables developers to systematically test context-aware applications in laboratory settings. To achieve this, TestAWARE is able to download, replay and emulate contextual data on either physical devices or emulators. To support both white-box and black-box testing, TestAWARE has been implemented as a novel structure with a mobile client and code library. In black-box testing scenarios, developers can manage data replay through the mobile client, without writing testing scripts or modifying the source code of the targeted application. In white-box testing scenarios, developers can manage data replay and test functional/non-functional properties of the targeted application by writing testing scripts using the code library. We evaluated TestAWARE by quantifying its maximal data replay speed, and by conducting a user study with 13 developers. We show that TestAWARE can overcome data synchronisation challenges, and found that PC-based emulators can replay data significantly faster than physical smartphones and tablets. The user study highlights the usefulness of TestAWARE in the systematic testing of mobile context-aware applications in laboratory settings.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing**; • **Software and its engineering → Software verification and validation**

Additional Key Words and Phrases: Sensors, context aware computing, machine learning, mobile interaction, mobile sensing

ACM Reference format:

Chu Luo, Miikka Kuutila, Simon Klakegg, Denzil Ferreira, Huber Flores, Jorge Goncalves, Mika Mäntylä, and Vassilis Kostakos. 2017. TestAWARE: A Laboratory-Oriented Testing Tool for Mobile Context-Aware Applications. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 3, Article 80 (September 2017), 29 pages.
DOI: 10.1145/3130945

Authors' addresses: C. Luo, J. Goncalves, V. Kostakos, University of Melbourne, VIC-3010 Parkville, Australia, email: CHUL3@student.unimelb.edu.au, {firstname.lastname}@unimelb.edu.au; M. Kuutila, S. Klakegg, D. Ferreira, M. Mäntylä, University of Oulu, Oulu 90014, Finland, email: {firstname.lastname}@oulu.fi; H. Flores, University of Helsinki, Helsinki 00014, Finland, email: huber.flores@helsinki.fi.

2474-9567/2017/9-ART80 $15.00

https://doi.org/10.1145/3130945

1 INTRODUCTION

Mobile devices such as smartphones and tablets now embed a variety of sensors (e.g., accelerometer, gyroscope, GPS and magnetometer) that can provide context-aware services to increasingly large user bases. Driven by the richness of heterogeneous data and machine learning algorithms, context-aware applications are becoming increasingly popular across a variety of domains, including quantified self [27], urban sensing and e-health. To simplify the development of mobile context-aware applications, researchers have built a significant number of context management frameworks such as AWARE [10], EmotionSense [23] and Sensor Data Collection Framework (SDCF) [4]. Although development has become easier, testing of such context-aware applications remains challenging. For instance, it is challenging to test such applications in laboratory settings, and therefore developers and researchers have to resort to expensive pilots with users [21,22].

There are several reasons why testing mobile context-aware applications remains challenging. First, due to the heterogeneity of contextual data and complexity of algorithms, static testing by reading the source code is impractical for developers to conduct efficiently. Second, it is time-consuming and expensive to robustly test mobile context-aware applications in realistic pilot studies, due to the potentially large number of context states that require testing (e.g., specific time, locations and user groups). Thirdly, certain types of contextual data, such as human falls or applications crashes, are too rare for applications to capture during testing [17].

To alleviate these problems, researchers have proposed several testing techniques and tools. For example, ContextViewer [6] is a visualisation and processing tool that helps developers understand the distribution of contextual data values from different sensors. KnowMe [9] is a tool that can replay online data on a PC, and send test cases to context-aware applications for processing. Similarly, MobiPlay [22] is a remote testing tool that allows developers to test mobile applications on a server using recorded sensor data such as GPS coordinates and acceleration.

Despite these efforts, developers may still struggle with the lack of suitable datasets for testing. To overcome the lack of such datasets, researchers have designed several techniques to manipulate contextual data for testing specific applications. For instance, CRASHDROID [30] is a specialised tool for reproducing and replaying bug reports of applications for testing. Similarly, to confirm whether mobile crowdsourcing applications trigger questions in the correct context, one proposed approach is the integration of real-time, historical and simulated data to construct the intended context [17]. Similar work [24] can create, test and simulate interactions across mobile devices and ubiquitous smart infrastructures such as smart home.

However, for systematic testing of mobile context-aware applications, we still lack a laboratory testing platform that can replay/emulate a rich variety of context from different data sources, or test functional/non-functional properties (e.g., power consumption, machine learning accuracy and processing speed). For instance, using any combination of extant tools, developers cannot replay online historical data to test applications on physical devices or emulators. Even if a combination of tools supports the testing of an application, developers have to conduct multiple rounds of testing using each tool. Most importantly, none of existing tools provide the support of white-box testing. Without such support, developers can hardly locate and fix bugs in the source code. To address this issue, we have developed TestAWARE, a laboratory-oriented tool for testing mobile context-aware applications. The tool can download, replay and construct contextual data from different sources on either physical devices or device emulators. Based on a novel architecture consisting of a mobile client and code library, it supports both black-box and white-box testing, and testing of functional/non-functional properties. Testers can conduct different types of testing, depending on their actual requirements and resources. In summary, our work makes the following contributions:

1. Conceptually, we identify a gap between existing testing tools and the requirements for laboratory-based testing of mobile context-aware applications. Existing tools raise a significant number of challenges which impede effective and efficient testing of mobile context-aware applications in laboratory settings.
2. To overcome the identified challenges, we propose TestAWARE, a laboratory-oriented testing tool that supports the testing of mobile context-aware applications by constructing the intended context

and examining functional/non-functional properties. Unlike prior work supporting only one type of testing, TestAWARE aims at a wide variety of mobile context-aware applications and testing scenarios, by incorporating heterogeneous data (i.e., sensory data, events and audio), multiple data sources (i.e., online, local and manipulated data), black-box/white-box testing, functional/non-functional property examination and the environments of device/emulator.

3. We quantify the maximal replay speed of sensory, event and audio data for testing scenarios where testers want to efficiently conduct a testing task with longitudinal datasets (e.g., maximising the replay speed for sensory data collected across multiple months or years). We show that our tool can take advantage of PC-based emulators to replay data significantly faster than actual smartphones and tablets.
4. We evaluate our tool in a user study with 13 professional mobile application developers. The results highlight the strengths of our tool for testing mobile context-aware applications in laboratory settings.

2 RELATED WORK

The development of context-aware computing systems was one of the key challenges of Ubiquitous Computing [25]. At present, there exists a plethora of context management frameworks to develop mobile context-aware applications, e.g., AWARE [10], EmotionSense [23] and SDCF [4]. However, we observe that testing techniques and tools for context-aware systems have lagged behind the state-of-the-art in context-aware computing technology [16]. Muccini et al. [21] identified several challenges of mobile context-awareness testing, e.g., rich data sources, lack of testing tools and hardware difference. Testing tools need data to reconstruct the context where context-aware applications operate. Just as understanding user intent without explicit user input is challenging [8], it remains difficult and expensive to test mobile context-aware applications without relevant contextual data and data management tools [17]. In this section, we first summarise relevant work that addresses these problems. Then we identify the gap between the state of arts and the need in testing mobile context-aware application.

2.1 Testing Mobile Context-Aware Applications

Typically, the manufacturers of mobile operating systems provide basic software development kits (SDK) to support testing. For example, Android has a testing tool called Monkey [3] which can generate user interface (UI) events and certain system-level events. Monkey is able to record emerging errors in the testing. Similarly, Android also offers an automated testing tool called Monkeyrunner [20] which allows developers to test applications without modification of the application source code.

Furthermore, developers and researchers have developed tools for testing more complex behaviours. For instance, the GUI crawler [1] can identify bugs by automatically simulating Android application executions. Similarly, Mosaic [13] is an Android-based record-and-replay tool to capture and replay user interactions across interfaces. Based on the AWARE [10] context-aware middleware, ContextViewer [6] is a visualisation and initial processing tool to help developers understand contextual datasets collected by AWARE. However, these testing tools, including industrial tools (e.g., Testdroid [14]), do not have the record-and-replay feature for contextual data.

Hence, a large body of work focuses on the features of creating or replaying context events. Amalfitano et al. [2] highlight that mobile applications can be tested by event-based techniques with the consideration of both context and UI events. Tonjes et al. [28] present a semi-automated method for test case generation and test case execution. Specific values of contextual data can be automatically created for each test case. RERAN [11] is an Android-based record-and-replay tool to collect low-level data streams such as UI events and hardware sensor data. Although it can record data from various Android sensors such as accelerometer, it is unable to capture GPS information because GPS is a stand-alone service in Android. KnowMe [9] is a tool that can replay data collected by AWARE. It allows testers to specify the speed of replay. However, it only works on PCs and can only fetch data from online databases of AWARE. MobiPlay [22] is a remote testing tool that allows developers to test Android applications using recorded datasets such as GPS and accelerometer.

MobiPlay runs these applications on a remote server. The MobiPlay client on mobile devices acts as the GUI of the targeted applications. A limitation of MobiPlay is that developers cannot use datasets of other middleware from online sources or device storage. Also, developers cannot launch tests without a specialised server.

If the intended context is uncommon for a mobile device to capture, generic testing or record-and-replay tools cannot provide effective support in testing. Hence, researchers have built a number of specialised tools. Griebe and Gruhn [12] present a model-based testing method that generates an appropriately selected group of test cases according to system design models. CRASHDROID [30] is a specialised tool to reproduce and replay bug reports for testing. Also, Roalter et al. [24] propose a development tool to create, test and simulate interactions between mobile devices and ubiquitous smart infrastructures such as smart home.

However, functional-testing tools and techniques which also support the testing of non-functional properties (e.g., energy consumption, machine learning accuracy and processing speed) of applications are non-existent. The feature of testing non-functional properties is crucial because mobile devices have limited computing resources. For example, it is necessary to balance power consumption and classification accuracy for mobile context-aware applications [7]. Although several specialised tools, such as the power estimation IDE (integrated development environment) [19] and FOREPOST [18], focus on performance, it is a significant burden for developers to test functionalities and each property of an application using multiple tools.

2.2 The Gap to the Need for Mobile Context-Aware Testing

Table 1. Comparison

Tool/Method	Data Type	Data Source	B/W Box Testing	Non-functional	Environment
Monkey [3]	Event	Script	B	-	D/E
Monkeyrunner [20]	Event	Script	B	-	D/E
GUI crawler [1]	Event	Script	B	-	D/E
Mosaic [13]	Event	Record	B	-	D/E
ContextViewer [6]	Sensor	Online	-	-	-
Testdroid [14]	Event	Script	B	-	D
[2]	Event	Script	B	-	D/E
[28]	Sensor, Event	Script	B	-	-
RERAN [11]	Sensor, Event	Record	B	-	D/E
KnowMe [9]	Sensor	Online	-	-	-
MobiPlay [22]	Sensor, Event	Record	B	-	-
[12]	Sensor, Event	Model	B	-	D/E
CRASHDROID [30]	Bug Report	Record	B	-	D/E
[24]	Event	Script	B	-	-
[7]	Sensor, Event	Online	-	ML, PS, PC	D/E
[19]	-	-	-	PC	-
FOREPOST [18]	Sensor, Event	Online	-	PS	D/E
TestAWARE	Sensor, Event, Audio	Online, Local, Script	B/W	ML, PS, PC	D/E

B – Black-Box Testing, W – White-Box Testing, D – Device, E – Emulator, ML – Machine Learning, PS – Processing Speed, PC –

Power Consumption

Summarising the capabilities of existing tools and methods, Table 1 frames TestAWARE in relation to the state of the art using a number of criteria:

1. *Data type*: mobile context-aware applications may collect and process various kinds of data, including hardware sensory data, software data, human input, audio and video [10]. We found that half of previous work only focused on a single data type. Also, replaying audio and video data during testing is rare in the literature. An ideal testing tool should support a wide range of data types to meet requirements of different applications.
2. *Data source*: test cases are based on testing data. For simple low-dimensional context, developers can easily generate and record ad-hoc data. For example, the battery charging status of a smartphone can be changed by connecting or disconnecting the USB cable on a computer. However, for rare or multi-dimensional context, developers may not have the opportunity to conduct systematic and exhaustive testing across the intended context. In these cases, developers can resort to re-using historical or manipulating data by writing a script. However, all existing tools only support one such way of obtaining testing data. This poses a crucial challenge for developers to reuse existing datasets. Even, some record-and-replay testing tools do not accept data from other sources. Therefore, a tool allowing multiple data sources can enable the testing of applications aiming at uncommon context and can significantly simplify the testing process.
3. *Black/white-box testing*: in the software testing process, developers normally use both two methods, black-box testing and white-box testing, to test an application. Many existing provide neither method, and the remainder of tools focus only on black-box testing, which can only provide functional feedback. Our search did not identify a single white-box testing tool for mobile context-aware applications, even though white-box testing is very important in software development. This is because black-box testing only verifies whether a software can achieve its functional objectives, but cannot ensure that all the possible states of software are valid. Using these tools alone, developers do not have any support to examine internal behaviours of their context-aware applications. Hence, providing both methods of testing is a necessary requirement for a testing tool.
4. *Non-functional testing*: non-functional properties play a crucial role in the practical use of applications. For mobile context-aware applications, testing tools should support 3 aspects which are closely related to user experience: machine learning performance, processing speed and power consumption [7]. Despite the existence of specialised testing tools, very few support this kind of testing.
5. *Environment*: during testing, developers may need to examine the states of applications that correctly operate on physical devices. Also, the measurement of processing speed requires the environment of physical devices. When developers do not have access to specific physical devices (e.g., having certain screen sizes or OS versions), they must rely on emulators. For this requirement, testing tools must support both device and emulator environments, rather than only executing tests on source code using a server or PC. Approximately half of existing tools and methods do not provide this feature.

Our analysis highlights the state of the art in mobile context-aware testing tools, and shows that there exist significant challenges in the testing of mobile context-aware applications, as also reported in literature [17]. We argue that the requirements for a testing tool for mobile context-aware applications are to support all the different testing scenarios from the criteria. The comparison across existing tools highlights a gap in the state of the art: none of them aim to work as a holistic tool for the complete testing process on diverse mobile context-aware applications and testing types. For instance, using any combination of existing tools, developers cannot use online historical data to test applications on physical devices or emulators. Even if a combination of tools supports the testing of an application, developers have to run multiple rounds of testing using each tool. Most importantly, prior work ignores the support of white-box testing. With existing tools, developers can hardly locate and fix bugs in the source code. Hence, our research addresses this gap in literature by developing TestAWARE to match the requirements set out in Table 1.

3 FUNCTIONALITY

Before describing the technical details of our system, we first present a high-level overview of how our system functions, and how it supports developers in testing mobile context-aware applications in laboratory settings. To simplify our description, and make it more concrete, we consider a scenario where a developer is testing a mobile application that performs real-time fall detection (i.e., the application detects whether the phone user falls down or not. It does not report a fall if the phone itself drops from the user.) on the phone. The application is programmed to send an email to a caregiver every time a fall event is detected by the phone.

The main challenge that the developer faces in testing this application is the effort that it takes to test it in realistic settings. Every time the application algorithm is tweaked or improved, new tests need to be conducted to ensure that the application works well in detecting fall events. Typically, a developer would compile a new version of their application, install it on a phone, and then conduct physical tests where they drop the phone under a variety of condition (e.g. drop from the hand, fall down with the phone in the pocket, etc.). This testing regime is also representative of the practices of researchers who develop context-aware applications.

TestAWARE provides developers the ability to fuse simulated, historical, and real-time data in their testing regime [17] (Fig. 1). These can be combined to "reconstruct" the indented context within which their application should be tested. This context can be recreated on a physical device, or a device emulator.

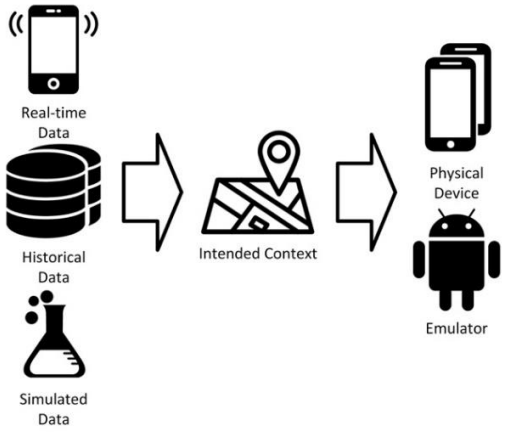


Fig. 1. TestAWARE supports the fusion [17] of real-time, historical, and simulated data during testing.

3.1 Context Data Preparation

The first step in using our tool, is that the developer needs to capture, download or generate contextual data. This data should reflect the conditions where the targeted application will be tested. In our scenario, the developer needs to record new data or download historical data from all relevant sensors, when the phone is dropped under a variety of circumstances. Therefore, the developer needs to orchestrate and record human fall events, as well as likely false-positive events (e.g., the phone dropping from a user's hand). Many kinds of context middleware, such as AWARE, can be used to record the relevant sensor data (e.g., accelerometer and gyroscope), and the data needs to be stored in an AWARE-compatible format (i.e., records with timestamps).

We note that developers can also create simulated sensor values, for example simulated accelerometer values or constant sensor values. This approach of using simulated data may be helpful in cases where rare or specific events need to be emulated. Developers can construct synthesised contextual data using the TestAWARE library. In our scenario, the developer could, for instance, create a for-loop that samples values

from a normal distribution and stores those values as simulated accelerometer values using the TestAWARE library.

3.2 Black-box Testing

To conduct black-box testing of the application, the next step will be to configure the TestAWARE client. The developer should install the client and the application in the testing environment (e.g., phone or emulator).

Next, the TestAWARE client needs data details before it replays data. To achieve this, the developer should indicate where the context data is stored (either online or in local storage). According to the AWARE-compatible data format, a single file or database table represents data for a single sensor. Therefore, the developer needs to indicate the database table or local file that contains the contextual data to be replayed, and then the TestAWARE client fetches all data to make sure it is available locally for the tests.

Subsequently, the developer needs to define in the TestAWARE client a replay task: this is a definition of what sensor data should be replayed, and the timestamp range. A replay task is analogous to a set of test cases. A developer may define multiple replay tasks: some replay tasks may include more sensors that others, and different replay tasks can use different context data files. For instance, in our scenario the developer could create one replay task that uses only the accelerometer values the developer has previously recorded; another replay task can use both accelerometer and gyroscope. Additionally, the developer could create a replay task that uses accelerometer and gyroscope values captured in a lab setting, while another replay task could be defined to use the values captured from a bedroom. If an application needs a sensor to operate, but that sensor is not included in a replay task, then the device uses data from the actual physical sensor.

Finally, the developer needs to begin execution of a replay task on the TestAWARE client, and define the replay speed. Once the replay has started, the developer switches to the targeted application, and observes/monitors its behaviour. In black-box testing we would assume that the application outputs results either into a database, or perhaps via debug messages. Once the replay task is finished, the developer needs to inspect the output of the application, and identify problems or errors. In our scenario, the developer could simply inspect the timestamps when his application detected a fall event.

3.3 White-box Testing

If the developer wishes to conduct white-box testing of the application, the next step will be to import the TestAWARE library into the application. The developer then needs to edit the source code of this application to define, configure, and control the replay tasks that are desired. Effectively, the developer needs to programmatically configure the TestAWARE client for data replay. The reason for programmatic configuration is to enhance automation, and facilitate repeatable tests. Ideally, this code would appear in the application when it is ready to start processing sensor values.

There are additional changes to the source code that the developer can make, to improve testing. First, the developer can add **assertions** that the TestAWARE library provides. These assertions will be communicated, at runtime, to the TestAWARE client, and help the developer identify bugs. In our scenario, the developer would add such an assertion after a fall is detected, and indicate: a brief description of the assertion (e.g. “React to fall”), the expected value (e.g. “Message to caregiver is sent”) and the actual value (e.g., “Message to caregiver is not sent”). This assertion helps the developer test whether the application responds as expected to context. After the tests are complete, the developer can inspect all assertions that were logged during the test, to identify problems in the behaviour of the application.

Second, TestAWARE allows source code modifications to facilitate the evaluation of applications to use **machine learning assertions**. This functionality is meant to help developers verify that modifications to their machine learning code have resulted in higher accuracy during the tests. The developer needs to make sure that the replayed data has ground-truth labels. These labels are created during the Data Preparation phase, and they need to reflect the desirable behaviour of the application. For example, in our scenario the developer would include “FALL” labels with the captured accelerometer values. Then, the developer would edit the source code that processes each new incoming accelerometer values, and add an assertion using the

TestAWARE library. The assertion would compare the output of the application's machine learning classification (e.g. "FALL" or "NO-FALL") versus the ground truth label that is included in the replayed data. In case where the context recognition is not classification but a regression, the assertion compares the expected value (ground truth label) versus the predicted value (as predicted by the application).

Third, the developer can insert a code snippet while the application launches, to enable **energy consumption profiling**. This code snippet informs the TestAWARE client of the sensor energy specifications of the handset that is tested. In our scenario, the developer would create a snippet that: defines the "Nexus 5" as the handset; indicates "Accelerometer" (one of the sensor names being replayed); indicates the sensing delay (e.g. "Normal", as specified by Android documentation); and indicates the expected power consumption per hour (e.g. "0.4 mAh per hour"). The expected power consumption may be defined according to the developer's expectations, or perhaps the hardware specifications of sensors (although those tend to be inaccurate).

Finally, the developer can perform **processing speed profiling** by calling functions provided by the TestAWARE library. These functions are meant to be called before and after a time-consuming operation takes place, such as regression or classification. The developer can assign each measurement a label so that multiple modules in the code can be measured without conflict. The functions are used to measure the time that it takes for the operation to complete, and they communicate these results, in real time, to the TestAWARE client. In our scenario, the developer would add this pair of statements around the line of code that initiates classification when new accelerometer sensor values arrive. The developer would then see in the TestAWARE client the average and worst-case durations recorded during the test.

4 ARCHITECTURE

TestAWARE consists of a mobile client and code library, as shown in Fig. 2. We will detail the functions, usage scenarios and design trade-offs of each component in the following subsections. Although TestAWARE aims at the testing only on the Android platform due to its popularity, this architecture can be deployed to build similar testing tools on other platforms, such as iOS and Windows.

As suggested by previous work [9,10], we implement both the mobile client and code library for the Android platform. The purpose of TestAWARE is to facilitate the testing of different mobile context-aware applications. Our main objective is to minimise the reliance on testing that requires the end-users of targeted applications. Conceptually, TestAWARE is able obtain and replay "context", and thus provide a reliable and repeatable setting for testing context-aware applications.

Before replaying contextual data, developers should collect relevant datasets either from online sources or from the device storage using local providers. Developers can also generate synthetic data programmatically using the data manipulator. Once contextual data is available, the developer can replay it using the data replayer, while in parallel the energy evaluator estimates the energy consumption of each sensor involved in the replay. Both the client user interface and command API (Application Programming Interface) of the code library provide replay control.

In black-box testing, developers cannot modify the targeted application. Hence, the targeted application receives and processes data, and remains ignorant of the presence of the TestAWARE client and code library. In this case, the targeted application outputs results as usual, and developers have to analyse these results through the functionality of the targeted application (e.g., monitoring the user interface, or inspecting the database of the targeted application), because the application does not send the results to the TestAWARE client.

TestAWARE allows developers to perform white-box testing by importing the code library into the targeted application. In this case, a context-aware module in the targeted application can output results to the result recorder of the TestAWARE client. Then the machine learning evaluator can generate performance analysis based on the recorded results. In addition, developers can make use of the processing speed evaluator to record the time of executions. Similar to other white-box testing methods, the limitation is that the testing requires modification in the source code of the targeted application.

TestAWARE's architecture entailed several design decisions and trade-offs. In turn, these were guided by the requirements we had set out for our tool. In Table 2 we describe how our requirements (from Table 1) were mapped to design choices in TestAWARE's architecture.

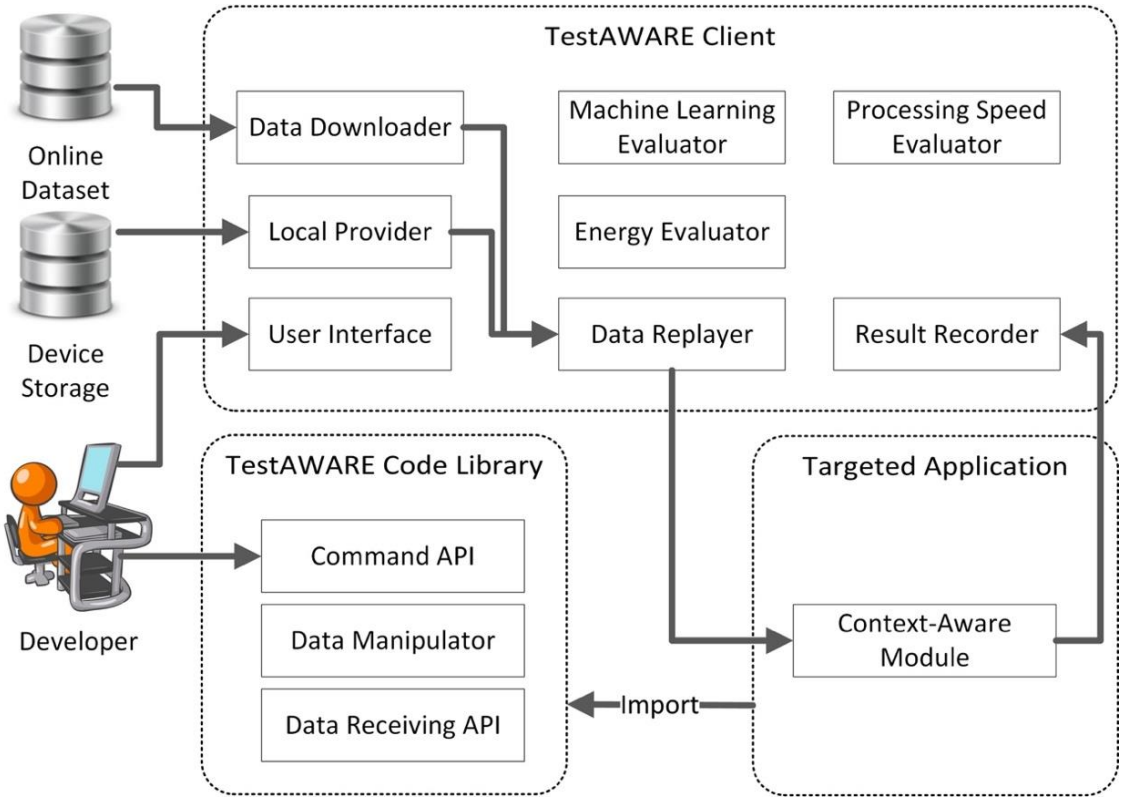


Fig. 2. TestAWARE architecture. The Client and Code Library are used to test the Targeted Application.

4.1 TestAWARE Client

The TestAWARE client is an Android application for mobile devices. It can also run on PC-based device emulators. During testing, the client runs simultaneously with the targeted application. The client user interface provides developers with testing data, replay data and read non-functional performance results. The client supports developers to conduct black-box testing on physical devices and emulators. Since developers cannot modify source code in the black-box testing, deploying a client running with the targeted application is the only viable approach. Fig. 3 shows screenshots of the client user interface during testing. Note that developers can also write scripts using the code library to automate the same tasks.

4.1.1 Data Downloader. The data downloader allows developers to test applications with data replayed by the device, using existing online datasets such as an AWARE database. For example, developers can use online datasets from previous projects or experiments. Specifically, developers can specify credentials for a database (including database host, username, password, data table and timestamp period). Thus, the data downloader can download the respective dataset completely or partly to the device according to the user specifications.

Table 2. Mapping from requirements to design choices.

Requirement	Design Choice
Support the replay of heterogeneous data	Incorporate support for sensory, event and audio data in data downloader, local provider, data manipulator and data replayer
Allow multiple sources for testing data	Support online data by data downloader; support local data from device storage by local provider; support manipulated data by data manipulator
Support black-box testing	Replay data using the inter-process communication (Android Intent) in line with data collection of applications
Support white-box testing	Provide APIs for commands and audio data transmission using a code library in Java's JAR package, which the targeted application can import
Enable non-functional testing	Include evaluators for machine learning performance, processing speed and power consumption
Support testing on physical devices and emulators	Deploy the client as an Android application that runs with the targeted application; replay the data with a suitable number of threads (*i.e.*, the number of available CPU cores)

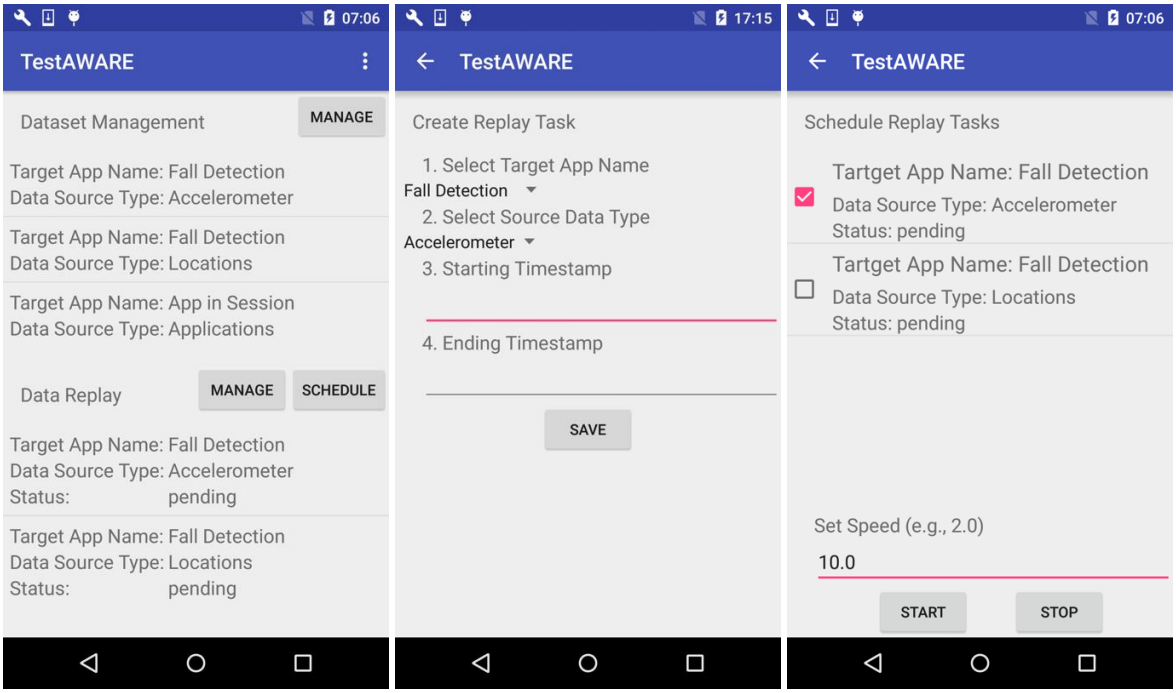


Fig. 3. The TestAWARE client UI can be used to manage datasets and replay tasks.

To replay online data, the data downloader first downloads the data to its local storage. In contrast, KnowMe [9] simply fetches online data on the fly during the replay. Although downloading all the data may take a long time, developers can reuse the downloaded data if they need to replay data again. Another advantage of downloading is that the replay of downloaded data can stay uninterrupted by network faults.

4.1.2 Local Provider. To test applications with data already recorded on the device, developers can use a local provider to include such data in the replay. A typical scenario can be that developers first record some data on the phone using the targeted application or a middleware such as AWARE, and then they replay the data in the testing with the help of the local provider. Similar to data downloading, developers need to input file path, table name and timestamp period.

In Android application development, there are two options to copy local data: using a content provider or file stream. However, Android requires reading and writing permissions for each specific content provider when the data-processing application (e.g., TestAWARE) is developed. We are not able to know the details of these content providers since they are different across each testing task. Comparatively, it is easier for applications to read and write local data using a file stream, because Android only asks for a general permission "<uses-permission android:name= "android.permission.WRITE_EXTERNAL_STORAGE" />". TestAWARE copies local data files to its own folder. Then developers can include the local data during testing.

ALGORITHM 1: Concurrent Data Replay

***INPUT**: a dataset with nonempty data sources in the replay D={d[0], d[1], …, d[n]}, speed multiple v*
BEGIN:

```
1:  foreach data source d[i] do in parallel
2:      finished[i] ← false
3:      if d[i] is audio then
4:          goto line 20
5:      else
6:          set data instance I[current] as the first data instance of d[i]
7:          send I[current]
8:          if d[i] has next instance I[next] then
9:              set t as the time difference between I[current] and I[next]
10:             wait t/v
11:             I[current] ← I[next]
12:             goto line 7
13:         else
14:             finished[i] ← true
15:             while for all integer j, -1<j<n+1, finished[j] is true do
16:                 goto END
17:             end while
18:         end if
19:     end if
20:     set frame F[current] as the first frame of d[i]
21:     send F[current] via each channel of d[i]
22:     if d[i] has next frame F[next] then
23:         set s as the sample rate of d[i]
24:         wait 1/(s×v) second
25:         F[current] ← F[next]
26:         goto line 21
27:     else
28:         goto line 14
29:     end if
30: end for
```

END

4.1.3 Data Replayer. Unlike prior work [9], the data replayer of TestAWARE only replays localised data (i.e., downloaded data or data recorded on the device) to avoid the effects of unstable network connection. Also, localised data can be reused in different testing tasks. To carefully examine software details or accelerate the testing for longitudinal datasets, developers can specify replay speeds that are slower or faster than the real-time clock. The data replayer concurrently replays data from each data source using a suitable number of threads (i.e., equal to the number of CPU cores in the current environment). To synchronise sensor values, events, and audio files during replay, we have developed a concurrent algorithm whose pseudocode is shown in Algorithm 1.

We have implemented the data replay module using the *java.util.concurrent* package provided by Java 8. TestAWARE schedules all the data sources selected by developers using a *ScheduledExecutorService*, which considers each data source as a scheduling task. To replay data at the speed set by developers, this service schedules tasks periodically by recalculating the timestamp using the predefined replay speed. To maximise the efficiency in replay, the service detects the number of cores on CPUs and spawns the same number of threads. Once a task replays all the data from a data source, the service will finish the task and give resources to other ongoing tasks. For audio data, TestAWARE accepts Waveform Audio File Format (WAV format) with one channel. Unlike other sensory data and events, audio data is a stream of frames. For example, the sample rate of WAV is normally 44100 Hz. In practical replay, we found that sending every sample per time is a significant burden (44100 executions per second) for devices or emulators. To solve this problem, we create a buffer to send every 441 samples per cycle, which means 100 executions per second. This strategy substantially reduces the computational cost for audio data replay.

4.1.4 Result Recorder. Allowing for modifications of the source code in a white-box testing regime, the targeted application can output results to the TestAWARE client. The result recorder collects and stores this output on the device for further analysis. Developers can insert code into the source code to verify the internal behaviours of the targeted applications. For example, a line of text output code can effectively test whether the program runs into a branch under specific conditions. Developers can also record debugging output information or assertions to compare the actual values and expected values in the white-box testing. This feature is useful when developers want to evaluate machine learning modules. The result recorder can capture each prediction from the machine learning algorithm, and each ground-truth value from one of the data sources. Then, the machine learning evaluator compares the two groups of outputs and generates the analysis results. Furthermore, the result recorder can record real-time output for the processing speed calculator to analyse the real-time performance of the targeted application.

4.1.5 Machine Learning Evaluator. The accuracy of context recognition is an important requirement in the development of context-aware applications. However, evaluating machine learning algorithms in these applications is non-trivial. First, there are many kinds of the learning problems involved in these applications, including binary classification, multiclass classification, regression, hard clustering, soft clustering and so on. Second, datasets of these applications may contain data labels with different frequency. This is a challenge in the analysis of machine learning performance.

Suppose a dataset contains N raw data instances X = {x1, ..., xN}. The machine learning algorithm may directly take these data instances as input, serving as a function $f: X \rightarrow Y$, where Y is the output space of classification or regression. If the dataset also contains the ground truth y_i in Y, corresponding to each raw data instance x_i, the machine learning evaluator can easily assess performance by comparing the output of the algorithm $\hat{Y}$ and the ground truth Y given by the dataset.

However, it is conceivable that a machine learning algorithm of a targeted application may filter raw data or transform it into another form (*e.g.*, using feature extraction methods) to construct input which is not actually in the historical dataset. Additionally, the dataset may only contain Y_{sub} which misses some values from the complete ground truth sequence Y (*i.e.*, $Y_{sub} \subsetneq Y$). In these cases, it is challenging to correctly match each ground truth value of the dataset with the output of the machine learning algorithm. Hence, assuming the ground truth appears later than the triggering context (*e.g.*, when using Experience Sampling Method [15], to collect ground truth labels), we propose an output matching algorithm which operates together with the

machine learning algorithm within the targeted application. Algorithm 2 shows the pseudocode of this algorithm. Note that developers have to indicate the maximal possible *time difference* between the last raw data instance and the ground truth label by specifying the delay tolerance T in seconds. Both $\hat{Y}$ and Y are saved by the result recorder.

ALGORITHM 2: Output Matching Algorithm

***INPUT**: machine learning algorithm f, a dataset with raw data instances X = {x[1], ..., x[N]} and ground truth sequence Y (it is uncertain whether this sequence is complete), empty sequence $\hat{Y}$, delay tolerance T in real-time clock*
***OUTPUT**: output value $\hat{Y}$, where $\hat{y}_i$ is considered to be corresponding to y_i in Y*
BEGIN:

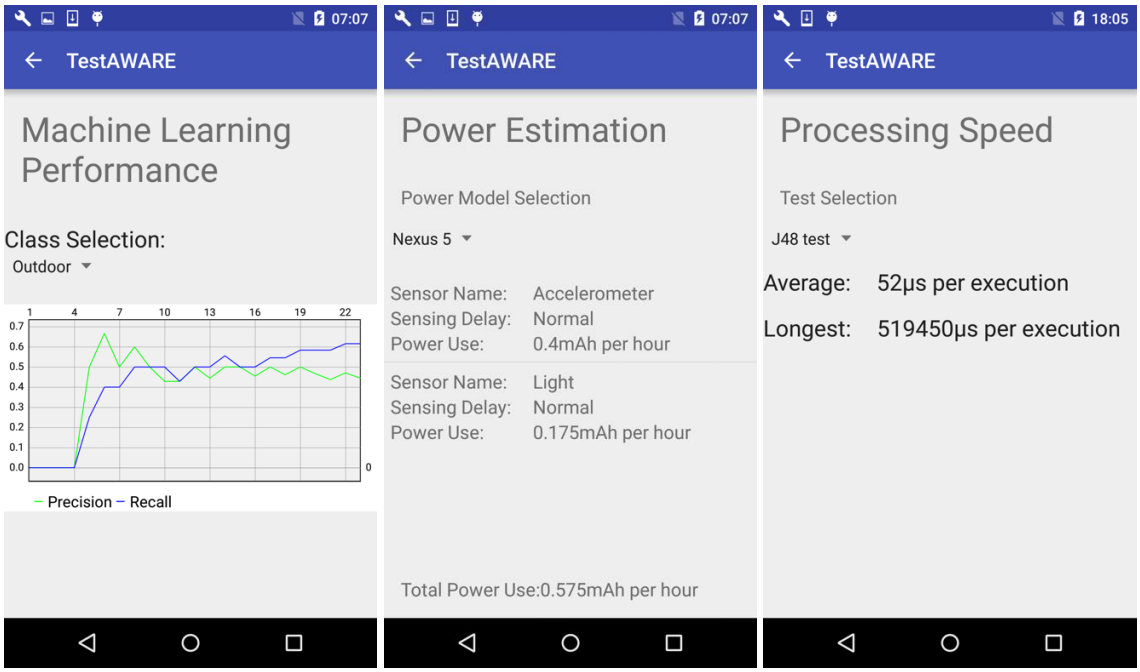
```
1:  while Y has next instance y[next] do
2:      set Ty as the timestamp of y[next]
3:      while X has next instance x[next] do
4:          set Tx as the timestamp of x[next]
5:          input x[next] into f
6:          if Tx > Ty then
7:              add NULL into Ŷ
8:              break
9:          end if
10:         if f generates output ŷ then
11:             if Ty − Tx < T then
12:                 add ŷ into Ŷ
13:                 break
14:             end if
15:         end if
16:     end while
17: end while
18: output Ŷ
```

END

For white-box testing, the machine learning evaluator uses the results captured by the result recorder to assesses accuracy. For classification tasks based on supervised or unsupervised learning algorithms, the machine learning evaluator measures prediction accuracy by comparing the output values to the ground truth in the dataset. For classification problems, the machine learning evaluator further measures the precision and recall of each class, as shown in Fig. 4a. For regression problems, the machine learning evaluator measures the mean absolute error (MAE) and mean squared error (MSE). The machine learning evaluator tracks performance during the whole period of data replay. Thus, developers can gain insights about the impact of increasing data amount on the machine learning performance.

4.1.6 Energy Evaluator. The energy evaluator, together with the machine learning evaluator, supports developers in balancing the tradeoff between power cost and the accuracy of context recognition. Similar to the sensor-trace method proposed in [19], developers can provide a sensor power model for a specific device. Based on this model, the energy evaluator estimates the power consumption by calculating the usage of each sensor with the sensing frequency in the replayed dataset. Fig. 4b shows an example of the output. If developers do not have such a model, they can refer to hardware manufacturers using AWARE.

4.1.7 Processing Speed Evaluator. For real-time sensing applications, computation time is a critical measure of an execution. For example, slow processing speed decreases the user experience of a context-aware UI. Hence, the TestAWARE library enables developers to measure and record the execution time of intensive tasks (e.g., machine learning) during white-box testing. Before and after the software module code in the targeted application, the developer can call related functions provided by the TestAWARE library. To measure multiple modules without conflicts, the developer can assign each measurement a label. After the test is complete, the TestAWARE client visualises the average and worst durations for the developer to view (Fig. 4c).



(a) (b) (c)

Fig. 4. Screenshots of the TestAWARE client showing: a) machine learning evaluation results. The developer can choose which class label is of interest (*e.g.*, "Outdoor"), and view the precision and recall performance (y-axis). The x-axis shows the sequence of ground truth labels during the test; b) the energy consumption profiling; c) the processing speed profiling.

4.2 TestAWARE Code Library

The TestAWARE code library is a Java JAR package. Developers can import this package into the targeted application for white-box testing and non-functional testing. They can also use this code library as standalone to generate and record simulated context. The code library provides significant flexibility during testing because developers can insert the code anywhere inside the targeted application, or even in the code of other applications. The only requirement is that the code from the code library can be executed in the same environment (i.e., device or emulator) as the TestAWARE client. However, a drawback is that developers without Java coding skills (e.g., front-end designers) cannot make use of functions in the code library.

4.2.1 Command API. The command API offers controls provided in the UI of the TestAWARE client, including downloading data, replay data and stopping replay. By writing simple code snippets using the command API, developers are able to automate the processes of dataset download and launch/stop data replaying in white-box testing. Compared to using the client in black-box testing, the command API is more suitable for large-scale testing involving a large number of devices or emulators. To make use of the command API, developers must import the code library into the targeted application.

During the initialisation of the targeted application, the application's source code and the commands written by developers are executed together by the device or emulator. Then the commands send messages to the TestAWARE client for the data replay. Thus, the targeted application can start to process the contextual data sent from the TestAWARE client.

To check the internal behaviours of the targeted application, developers can write output commands and assertions inside the source code, for example, a loop or a branch. Output and assertion results are recorded by the result recorder, as illustrated before.

In non-functional testing, the command API provides commands to record machine learning predictions and the execution time in nanoseconds. Also, to estimate power consumption, developers can use the related commands to build a power model for a device.

4.2.2 Data Manipulator. The data manipulator enables developers to quickly simulate intended context (e.g., a fabricated application crash or battery low event) by generating synthetic (i.e., simulated) data. Except audio data, developers can manipulate any type of sensory data and Android event, using the functions provided in the data manipulator. Then, the data manipulator creates and stores these values in the local storage of TestAWARE on the device or emulator. In either black-box or white-box testing, the TestAWARE client can replay this synthetic data in the same way it replays historical datasets in local providers.

Using the data manipulator, developers can efficiently simulate their intended context without recording values from a realistic context or obtaining a historical dataset. For example, healthcare application developers can create a set of fake GPS coordinates to test whether their applications can find the nearest hospitals.

Unlike previous work [12] generating values using a specific tool, the data manipulator is a chunk of reusable Java code. This means that developers can conduct the simulation of intended context at any time in the testing. Developers may use the data manipulator to generate some new contextual data depending on the initial output of the targeted application. For example, when a smart home application detects that the user is coming home, developers can simulate different indoor temperatures to test whether the air-conditioner is turned on.

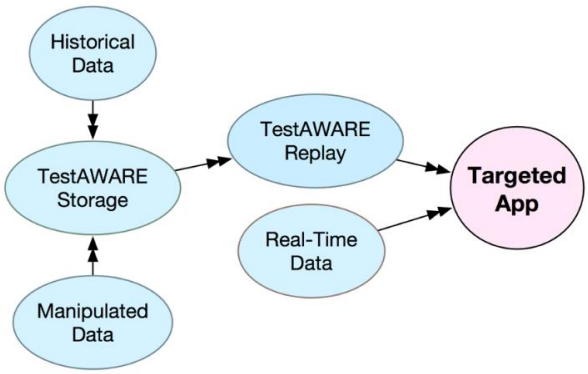


Fig. 5. Fusion of real-time, historical and manipulated data using TestAWARE in the testing.

4.2.3 Data Receiving API. The data receiving API is only for audio data replay. For AWARE-compatible hardware sensory data and raw Android events, the data collection and transmission process of the target application bears no difference to real-world scenarios. However, due to the specificity of microphone data stream, TestAWARE cannot override the microphone hardware to replay audio data. Consequently, the targeted application can only receive replayed audio streams using the data receiving API of the code library in white-box testing. Based on the data receiving API, developers can include audio as a data source in the replay to construct the intended context during testing.

4.3 Data Fusion Support

TestAWARE is able to replay datasets in the formats of AWARE-compatible hardware sensory data, raw Android events and audio files. Although developers can replay historical data to test applications processing different types of data, it may be still difficult to construct the intended context based on historical data only [17]. Fusing real-time, historical and simulated data is an effective way to construct context in more cases [17].

In TestAWARE, we implemented this idea for the construction of more sophisticated context, as shown in Fig. 5. Because TestAWARE does not change the way that the targeted application collects data, the targeted

application still considers replayed data as real-time data. Thus, the targeted application can simultaneously receive data from TestAWARE and hardware sensors. If developers want to test applications using manipulated data, they can create and store such data before testing. Since manipulated data is replayed by the data replayer during testing, TestAWARE supports the fusion of the manipulated, historical and real-time data.

Hence, developers can leverage data fusion to construct sophisticated context by combining replayed data and real-time data. For example, to test a battery monitoring application that recommends the closest phone-charging location depending on the current location, developers can replay a fabricated battery low event and use the actual (i.e., real-time) location coordinates such as GPS. Note that developers should set the data replay speed to real clock (i.e., 1x speed) if they include real-time data into the data fusion.

5 EVALUATION

For the purposes of evaluation, we have decided to use both quantitative and qualitative measures. We first discuss the deployment of the output matching algorithm, and then we quantify the maximal speed of data replay. Finally, we evaluate the usefulness of TestAWARE in a user study with 13 professional mobile application developers.

5.1 Selecting Delay Tolerance

The key aspect of our tool is its data synchronisation during replay, and it is especially important to assess how this works with "messy" real-world data and labels. Our output matching algorithm requires a parameter delay tolerance T to match each machine learning output and the corresponding ground truth label. Developers have to specify this tolerance empirically in their code snippet, depending on the characteristics of the dataset and the process of context recognition.

For example, in the field study in [29], the machine learning problem is to predict, every time the user unlocks their phone, whether the user is going to start a new task or continue their previous task. In this dataset, the ground truth labels ("NEW_TASK" vs. "CONTINUE_TASK") were collected by explicitly asking the users after they had unlocked their phone. As a result, the ground truth labels appear after the actual unlocking event, and this delay varies considerably due to inconsistent human labelling (Fig. 6). Hence, the ground truth labels have imprecise timing in this case.

In addition, the performance of the classifier (i.e., a constant classifier) during our testing can vary. For instance, we measured the time needed for our classifier to determine whether a user who just unlocked the phone intends to start a new task, or continue a previous task. The time needed by our classifier is shown in Fig. 7. This graph was constructed by visualising the values recorded by TestAWARE.

By comparing Fig. 6 and 7 we find that there is a discrepancy between when the ground truth label appears in the historical data, and when our classifier is able to validate its machine learning assertion during testing. We do not want to penalise the application for getting the timing of its prediction wrong, and therefore our Algorithm 2 allows for a Delay Tolerance to account for this discrepancy. In this case, we find that our classifier can take up to 0.678 ms to execute. At the same time, we find that some ground truth labels appear with delay of up to 100 seconds. Therefore, to correctly match our application's machine learning output and the ground truth labels, we would set the delay tolerance to be slightly longer than the majority (e.g. 95%) of the observed time difference between our application's machine learning output and the ground truth labels. This is approximately 5 seconds in our case.

5.2 Maximal Data Replay Speed

In the testing of context-aware applications with longitudinal data collection (e.g., chronic disease tracking), it is necessary for developers to replay a large amount of historical data in a short period. Hence, we quantify how fast TestAWARE can replay data. We selected 6 smartphones, 6 tablets and 4 PC-based emulators to replay sensor, event and audio data at the fastest speed. In the sensor data replay, we used one thread to

replay 10000 accelerometer readings, which is equivalent to about 1 minute of sensing with the highest fidelity. In the event data replay, we used one thread to replay 10000 ACTION_BATTERY_LOW events, which typically occur when the phone battery becomes low. For audio replay, we used one thread to replay a file in WAV format with one channel of 16-bit stream (sampling rate = 44.1 kHz, samples = 169529220).

5.2.1 Data Replay on Smartphone. First, we investigate the maximal replay speed on 6 off-the-shelf smartphones: LG Nexus 5 with Android 5.1.1, Samsung S6 Edge with Android 6.0.1, Motorola Moto G2 with Android 5.0.2, 2 × Motorola Moto G1 with Android 4.4.4 (G1-1 and G1-2), Motorola Moto G1 with Android 5.1 (G1-3).

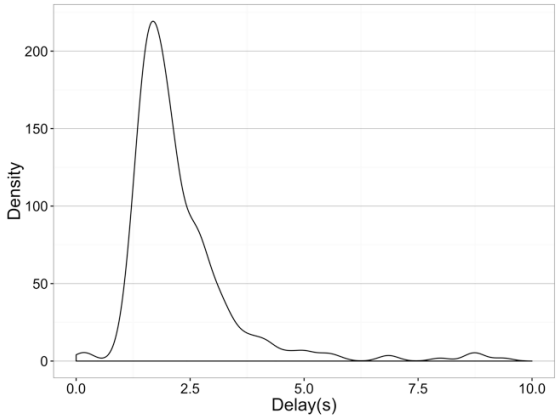


Fig. 6. Delay between unlock event, and ground truth label in our historical data.

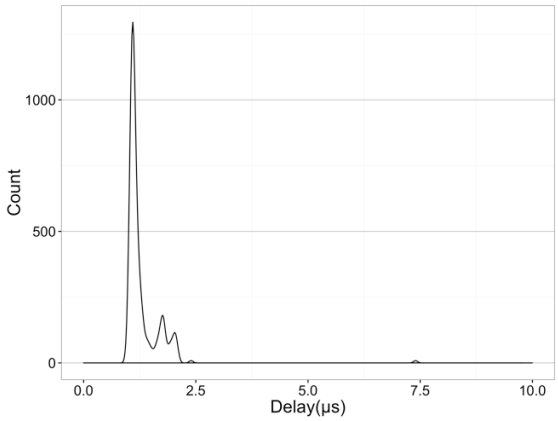


Fig. 7. The variation in the time needed by our classifier to make an inference.

Fig. 8 shows the maximal replay speed of audio on smartphones, in multiples compared to the original playback speed of the audio file. Results show that all these devices performed very differently. They were able to accelerate the replay speed to the range of 4.46 - 13.77 multiples. The best performance (13.77 on Nexus 5) was approximately triple the worst (4.46 on S6).

Fig. 9 depicts the maximal replay speed of sensor and event data on smartphones. Similar to audio replay, results show that devices performed very differently. All the handsets can replay at least 1000 instances of

either sensor or event data per second. Also, we observe that all the handsets replay events faster than sensor data. Except S6, the replay speed of events is significantly higher than that of sensor data.

5.2.2 Data Replay on Tablets. Next, we measured the maximal replay speed on 6 off-the-shelf tablets: Samsung Galaxy Tab Pro 8.4 with Android 4.4.2, 3 × Samsung Galaxy Tab4 10.1 with Android 5.0.2 (10.1-1, 10.1-2 and 10.1-3), 2 × Lenovo TAB3 7 with Android 6.0 (TAB 3 7-1 and TAB 3 7-2).

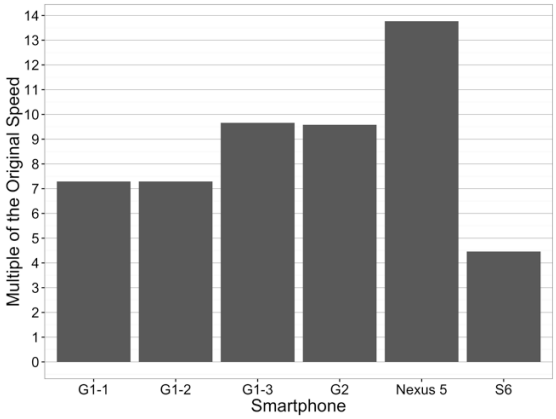


Fig. 8. Maximal speed of replaying audio files on smartphones.

Fig. 9. Maximal speed of replaying sensor data and events on smartphones.

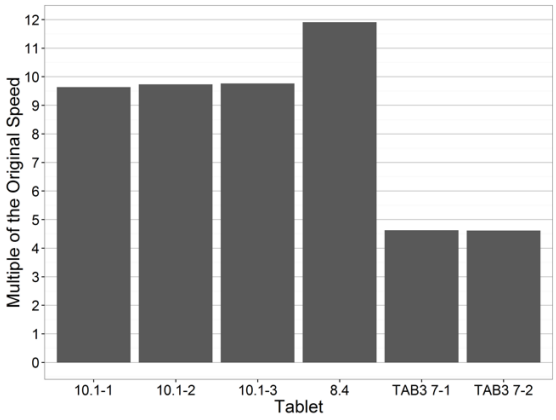


Fig. 10. Maximal speed of replaying audio files on tablets.

Fig. 11. Maximal speed of replaying sensor data and events on tablets.

Fig. 10 shows the maximal replay speed of audio on tablets, in multiples compared to the original playback speed of the audio file. Results show that the performance significantly differed. These tablets were able to replay the audio file within the speed range of 4.61 - 11.91 multiples. Tab Pro 8.4 performed the best with 11.91 multiples. Identical models of Tab4 10.1 and TAB3 7 had similar performance.

Fig. 11 presents the maximal replay speed of sensor and event data on tablets. Results show that the performance of replaying sensor data and events greatly varied among different models of tablets. TAB3 7 had the lowest performance with less than 700 instances of sensor data per second and 900 event instances per

second. On other models, the performance was more than doubled: at least 1800 instances of sensor data per second and at least 1900 event instances per second.

5.2.3 Data Replay on Emulators. Finally, we measured the replay speed on a smartphone emulator (Nexus 5 with API 22) and a tablet emulator (Nexus 10 with API 22) using 2 PCs: Mac Mini with Intel i7 2.3GHz and MacBook Pro with Intel i5 2.7GHz. We allocated 2GB RAM, 500MB Android VM heap size and only one CPU to each emulator.

Fig. 12 shows the maximal replay speed of audio on emulators, in multiples compared to the original playback speed of the audio file. Results show that all these emulators performed similarly. They achieved the maximal replay speed to the range of 19.31 - 21.88 multiples. The best performance (21.88 on Nexus 10 MacBook) was slightly better than the worst (19.31 on Nexus 5 Mini).

Fig. 13 presents the maximal replay speed of sensor and event data on emulators. Results show that all the emulators can replay at least 3000 instances of sensor data per second. For the replay of events, all the emulators can achieve a much higher speed: over 6000 instances per second. Similar to audio replay, the performance of replaying sensor data and events slightly varied across emulator types (smartphone and tablet) and PC models.

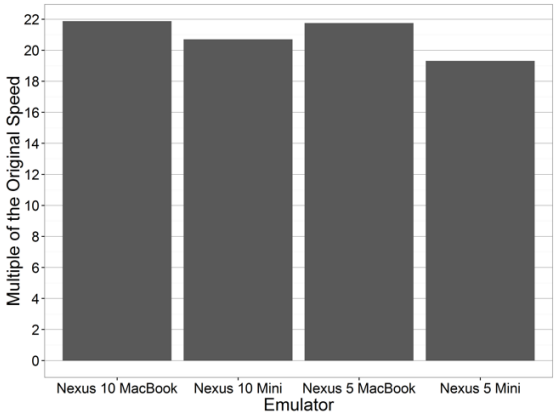


Fig. 12. Maximal speed of replaying audio files on emulators.

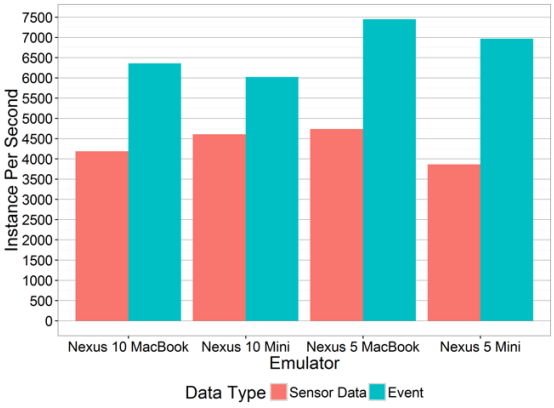


Fig. 13. Maximal speed of replaying sensor data and events on emulators.

5.3 User Study

To investigate the usefulness of TestAWARE, we conducted a user study with two real-world testing tasks. To select a suitable sample size of participants, widely accepted practices suggest 12 to 16 users [5]. Hence, we recruited 13 participants (3 females, 10 males, average age 25.8, average professional years 2.2). They were recruited through social media. All participants are professional programmers working in the industry of mobile software development for at least 1 year. The participants at least hold a bachelor's degree in computer science or related areas. Each participant was compensated with a gift card equivalent to 10 EUR.

All participants were interviewed individually in our lab. For each interview, we first introduced TestAWARE to the participant. We described and explained the features of the mobile client on a smartphone and an emulator running on a PC. Then we showed them a demonstration where an Android application receives replayed data from the TestAWARE client both on the phone and emulator.

Next, we completed a hands-on training session. Participants were instructed on how to use all the features of the TestAWARE client and code library. They were provided with available datasets, a template Android application project, API syntax and code examples. This training session was completed within one hour.

After the training was complete, participants were given a real Android application project for two testing tasks. The targeted application collects sensor data to predict applications to be used, and is implemented based on the work described in [26]. The predictions are shown on the screen as application icons. Before the study, we tested the implemented application carefully. Then we intentionally inserted a bug to the source code as a functionality flaw. Regardless of predictions, this bug always wrongly displayed a group of icons. The first testing task was black-box testing. The functionality of this application was introduced to participants. Participants were not familiar with the application's inner implementation structure and details. Participants were asked to test the application using the TestAWARE client only. The second testing task was white-box testing. Full source code and reference publication [26] were given to participants. In this task, participants were allowed to use the TestAWARE code library with the client. Participants were asked to not only test the application functionality, but also assess its non-functional properties including machine learning performance, power consumption and processing speed. Participants were given unlimited time to finish the tasks.

After both testing tasks were completed, each participant answered a questionnaire with nine questions relating to the testing tasks and our tool. For each question, participants were instructed to give a score on a scale from 0 (strong disagreement) to 100 (strong agreement). During the subsequent interview, we also asked each participant to explain their scores.

5.3.1 Results. The results in Table 3 show the numbers of participants according to their states towards the flaw for each stage. After black-box testing, 7 developers reported the flaw about icons because they found that the icons did not change during the whole process of data replay. However, 6 developers could not report this flaw because they thought that, even if the icons did not change, they may reflect correct predictions corresponding to the data replay. After white-box testing, all developers reported the flaw because they used run-time assertions to check the details of the program. Based on the results of assertions, they all correctly located the bug in the source code.

Table 3. Results of two testing tasks.

Number of Participants	Before Testing	After Black-Box Testing	After White-Box Testing
Reporting the flaw	0	7	13
Not reporting the flaw	13	6	0

We compared the states before testing and after black-box testing using Pearson's chi-squared test. We found that the black-box testing with the mobile client offered a significant help for testers to spot a flaw (P=0.008). In addition, we compared the effectiveness of our tool in the black-box and white-box testing. The results indicate that participants received a significantly greater support when using the TestAWARE client and code library in the white-box testing (P=0.02).

Question 1: Does data replay from TestAWARE help the testing in the two tasks? (average score: 90.8, median score: 90, standard deviation: 6.7) This question investigated whether our tool, in a practical manner, helps developers conduct both black-box and white-box testing of context-aware applications with data replay. The scores indicate that the participants largely perceived the necessity of data replay in the testing of context-aware applications, and that they identified a lack of tools to conduct data replay for this kind of testing. All participants considered data replay as a workable way to simplify the testing by avoiding collecting new samples of context in every round of testing. They also had unique opinions. In terms of testing cost, P2 stated: *"Data replay reduces the cost of data collection in industrial testing, without loss of fidelity*". Highlighting the support for automation, P5 stated: *"To automate the testing, testers must have a tool for data replay"*. Regarding the likelihood to identify a bug, P11 and P13 commented that data replay can reproduce bugs to help testers understand the causes: *"Without data replay, it is hard to reproduce a bug which was already detected in the past. If a detected bug cannot be reproduced, developers can hardly find the reason and fix the bug"* (P13). The low standard deviation shows that the participants had a little divergence when judging

the magnitude of usefulness. P8 giving a score of 80 stated: *"Data replay is able to make the application run in testing. Testing it directly in real-world context can also be a way to go"*.

Question 2: Does the TestAWARE mobile client help the testing in the first task? (average score: 80.2, median score: 85, standard deviation: 15.1) This question was to check whether the TestAWARE mobile client supports developers to conduct the black-box testing. The scores show that the participants considered the mobile client as a useful component of our testing tool. The majority of participants commented that the client is easy to use and does not require testers to write scripts. However, the high standard deviation indicates that participants judge its usefulness differently. Regarding the search for bugs, P1 stated: *"The client only replays the data for the tested application. It is hard to detect a bug because testers do not know whether the data is successfully received. And testers do not know the correct output from the tested application"*. In terms of testing coverage, P13 stated: *"Testers do not know how completely the application is tested. A number of bugs may be missed"*. P2, P5 and P11 stated: *"It does not help you locate the bugs"*. P12 stated: *"It helps testers who do not write code. But many bugs can be missed"*.

Question 3: Does the TestAWARE code library help the testing in the second task? (average score: 91.2, median score: 90, standard deviation: 7.5) This question was to investigate whether the TestAWARE code library supports developers to conduct the white-box testing. The scores show that participants perceived the substantial usefulness of the code library, with little divergence indicated by the low standard deviation. Most participants commented that the code library enables the automation and more careful examination in the testing. P1 stated: *"The commands can automate a large number of testing rounds"*. P8 stated: *"The code library helps the testing on not only high levels, but also the level of unit testing and integration testing. It can examine every step of programs. It also locates the bug in the code"*. P6 stated: *"It is easy to debug when a flaw is detected using the code library"*. P12 stated: *"The code library allows testers to match the output of software and different input"*. However, they argued that it takes long time to conduct testing using the code library. P8 stated: *"The limitation is that testers use a lot of time to write testing scripts"*. Accordingly, P13 suggested a solution: *"In practice, we may test only important components using this way, rather than all details"*.

To confirm the effectiveness difference of our tool in the black-box and white-box testing, we compared the scores of Q2 and Q3. First, we tested the normality of each distribution using the popular Shapiro-Wilk test. We found that the scores of Q2 are not normally distributed (P=0.010), but those of Q3 are (P=0.070). Hence, we used Mann–Whitney U test to verify the difference. The test identified a significant difference between the two sets of scores (P=0.041).

Question 4: Does the assertion function help the testing in the second task? (average score: 89.5, median score: 90, standard deviation: 10.1) This question was to investigate whether the run-time assertion function of the TestAWARE code library provides help in the white-box testing. The high scores indicate that the participants appreciated the importance of assertions in the testing. All participants agreed that assertions accurately located the bug in the code. P2 stated: *"Using run-time assertions is efficient and accurate. They are better than breakpoints because they require applications to run only once"*. However, the slightly high standard deviation indicated some limitations of run-time assertions: *"Given a bug location, it still requires some logical analysis to fix the bug"* (P10); *"Although a bug is located, understanding the whole scenario causing the bug is sometimes a complex task"* (P13).

Question 5: Does the machine learning evaluator help the testing in the second task? (average score: 82.7, median score: 90, standard deviation: 19.4) This question aimed to investigate whether the machine learning evaluator of our tool helps developers in analysing the machine learning performance of the application. The scores reveal a general endorsement among participants. P1 stated: *"It quickly gives an initial performance summary to testers"*. P11 stated: *"It reflects the quality of code which uses machine learning algorithms"*. Comparatively, some participants perceived only limited usefulness, causing the high standard deviation: *"The tool evaluates machine learning using several simple measures. It may produce inaccurate evaluation results"* (P6); *"Common testers may not pay much attention to machine learning performance. Also, users may not care the accuracy of machine learning results"* (P12).

Question 6: Does the power estimator help the testing in the second task? (average score: 83.2, median score: 90, standard deviation: 23.2) This question was to investigate whether the power estimator

helps developers to assess the power consumption. The scores show that the participants have the need to estimate the power consumption of context-aware applications. Regarding industrial mobile application development, P1 stated: *"The assessment of power use is indeed a process in mobile software production. An estimation is useful for testers to refer to"*. P10 stated: *"Estimation of power use from different sensors can help testers balance the data collection and battery preservation"*. However, several participants argued that the estimation may have large errors, resulting in the high standard deviation. P5 and P7 stated: *"The tool has only estimation. It cannot measure the actual power consumption of practical usage"*.

Question 7: Does the processing speed evaluator help the testing in the second task? (average score: 86.3, median score: 90, standard deviation: 10.4) This question aimed to investigate whether the processing speed evaluator helps developers in analysing the efficiency of certain procedures in the application. The scores reveal the considerable usefulness perceived by participants. P7 stated: *"Response time is an essential index of user experience for mobile applications, especially for Android"*. In terms of optimisation of applications, P6 stated: *"It helps testers find the bottlenecks which testers may try to optimise"*. P11 stated: *"Testers can compare different implementations and find the best one"*.

Question 8: Is the maximal replay speed of audio, sensor and event data sufficient to in the testing? (average score: 94.8, median score: 98, standard deviation: 5.7) This question was to investigate whether the maximal speed of data replay satisfies the need of developers in the testing. As indicated by the high scores and low standard deviation, all the participants agreed that the maximal replay speed suffices for efficient testing. P3 stated: *"It is fast enough. It reduces the testing time compared to testing in the real context"*. Regarding testing applications with longitudinal data collection, P11 and P12 stated: *"The high speed is enough to quickly complete a test if the data was collected across a long time"*.

Question 9: Is it a useful feature of TestAWARE to support both physical device and emulator? (average score: 96.9, median score: 100, standard deviation: 6.0) This question was to investigate whether TestAWARE helps developers in the testing by supporting both physical device and emulator. All participants agreed to the usefulness of the environment support, producing the high scores and low standard deviation. In terms of processing speed measurement, P1 and P4 stated: *"Using real devices to measure process speed is reliable"*. Regarding the convenience of development and testing, P6 stated: *"I like to use emulator for development and testing due to the convenience"*. Similarly, P8 stated: *"Emulators are easy for testers to automate testing"*. In terms of compatibility testing, P11 and P13 stated: *"It is necessary to test applications on real devices for checking compatibility on different hardware and OS"*.

6 DISCUSSION

Our overarching goal is to enable developers to test context-aware applications in laboratory settings. There are many reasons why this goal is desirable, including the ability to more systematically test the software, to enhance the reproducibility and replicability of test results, and primarily to reduce costs associated with participant-driven trials. Our tool enables developers to conduct automated tests in laboratory settings, thus greatly reducing the need (or offsetting the need) for user trials, at least in early stages of development of context-aware software. Clearly, we do not suggest that user trials are not needed, especially in terms of user interface design, but certainly automated testing can help in identifying functional and non-functional flaws.

6.1 User Study Findings

We begin by summarising the findings from our interviews with developers. The interviews first sought to investigate whether our tool can practically help developers test context-aware applications with data replay. Participants confirmed the necessity of data replay in the testing of context-aware applications, and they confirmed the lack of tools to conduct data replay for this kind of testing. Specifically, participants highlighted the simplicity of conducting multiple rounds of testing with TestAWARE. They also considered that data replay is effective in cost reduction, automation and bug reproduction. We also inquired the usefulness of the TestAWARE client in the black-box testing. The participants considered the client as a useful component because it is easy to use and does not require testers to write scripts. Additionally, participants considered the

code library as a useful component of the testing tool because of automation support, bug localisation and careful examination from low levels to high levels.

Regarding the testing of non-functional requirements, the participants felt that our tool supported them effectively for the analysis of machine learning performance, power consumption and processing speed. However, some participants argued that these three aspects are of different usefulness to them. They considered that processing speed is more useful than machine learning performance and power consumption estimation, due to the importance of user experience. In terms of the maximal speed of data replay, all participants agreed that the speed is sufficient for testers to conduct testing efficiently. Finally, all participants noted that support for both physical devices and emulated devices is crucial for testing. All the participants preferred the freedom to select testing environments between the two options. Some stated that, for convenience, they would only use emulator if they have data for replay. Some thought that they would always include physical devices in testing, not only for measuring processing speed, but also for the assurance of software compatibility.

To quantitatively evaluate our tool, we first statistically compared the effectiveness of our tool in the black-box and white-box testing. From the results of two testing tasks, we observed that our tool successfully supported testers in both the black-box and white-box testing. Furthermore, testers receive significantly greater help when the code library was used with the client. From the interviews, we confirmed this observation by the significant difference in the score distributions from the two related questions. Within the white-box testing support, the scores show that the assertion is a critical and useful feature of our tool. From the scores of the remaining questions, we confirmed the necessity of data replay and testing non-functional properties including machine learning performance, power consumption estimation and processing speed. The scores show that processing speed is of the highest usefulness among non-functional properties. Regarding the maximal speed of data replay, the scores indicate that testers can conduct testing efficiently with adequate replay speed. Regarding testing environments, the scores show that testers need the flexibility across real devices and emulators.

6.2 Data Replay for Testing

By replaying heterogeneous data including AWARE-compatible hardware sensors, Android events and audio files, developers can use TestAWARE to test the context-aware modules of mobile applications. The results of our user study suggest that data replay using TestAWARE simplifies the testing of context-aware applications. Compared to prior work such as MobiPlay and KnowMe, TestAWARE supports audio from microphone as an additional data source. Furthermore, TestAWARE allows developers to construct the intended context by creating simulated/synthetic data, or using data fusion to combine real-time data and replayed data. This is a key characteristic that can reduce the efforts and cost in data collection or dataset download. For example, manipulated data can be used to construct uncommon contexts, such as app crashes and human falls.

When replaying data using TestAWARE, developers can set a replay speed that is faster or slower than the real-time clock. This can accelerate the testing with longitudinal datasets. Also, by replaying data at a slow speed, developers can carefully verify the execution details of context-aware modules, such as step counter. If the data fusion uses a real-time data source, the replay speed must be set to 1 (i.e. real-time).

TestAWARE enables developers to manage data replay using either its mobile client (i.e., black-box testing) or the code library (i.e., white-box testing). As indicated by our interviews, the mobile client plays a crucial role in the testing when testers do not have programming skills. Challenging as programming is, the code library enables automated testing and allows developers to check low-level details of software code by recording the output of applications. In addition, the code library provides the only access to the testing of non-functional properties such as machine learning performance, energy consumption and processing speed.

Finally, TestAWARE can operate on either physical devices or device emulators. This offers developers the freedom to select suitable environments for the testing. As suggested during our interviews, given access to data for replay, it is convenient to use solely an emulator in testing. However, for the assurance of software compatibility, it is necessary to run applications and TestAWARE on physical devices. Also, if developers

conduct tests with data fusion involving both replayed data and real-time data, they have to use physical devices and set the data replay speed to 1 (i.e., as fast as the real-time clock). According to our results, the maximal data replay speed of audio, sensor and event data on emulators is significantly faster than that on physical devices. This means that emulators are more suitable than physical devices when replaying data of longitudinal datasets.

6.3 Testing Non-functional Properties

By modifying the application source code in white-box testing, developers can use TestAWARE to record application output for debugging as well as testing non-functional properties, which is an important task for mobile applications due to the limited computational resources and battery life on mobile devices [7]. TestAWARE provides an analysis of the machine learning performance, power consumption and processing speed of the targeted application. Developers can make of these features to optimise applications for non-functional requirements.

As indicated in our interviews and prior work [19], these properties may be of different importance to specific applications. For example, compared to processing speed, power consumption is not an important measure for an application processing audio since developers normally cannot change the configuration of the microphone. It is worth noting that TestAWARE does not force developers to write testing code for all features. According to the mechanisms of their context-aware applications, developers can include or exclude specific modules from the code library when writing the code for the white-box testing.

6.4 Implications for Testing Mobile Context-Aware Applications

This section summarises the impacts of our work to the testing of mobile context-aware applications, in terms of conducting effective testing using our tool (i.e., guidelines for testers) and designing similar testing tools (i.e., guidelines for testing tool developers/researchers).

6.4.1 Heterogeneous Data. Some mobile context-aware applications rely on only one kind of contextual data. Testers can easily find tools listed in Table 1 to replay or manipulate sensory data or events. For mobile context-aware applications collecting heterogeneous data, testers have much fewer selections. To test mobile applications based on audio data, testers can only choose TestAWARE. Unlike sensory data and events, common testers cannot write scripts to generate audio files for testing, since they are not human-readable. Yet, to test mobile context-aware applications, TestAWARE is the only tool supporting sensory data, events and audio, enabling a new type of testing which is not covered by prior work. Testers can set a speed for data replay, conducting tests which are faster (e.g., to save time when testers use a longitudinal dataset) or slower (e.g., to carefully monitor the applications) than the real-time clock.

However, testers should have a suitable dataset for TestAWARE to replay. If testers are able to obtain such a dataset (e.g., by collecting application usage data from a sufficient number of users), tools, such as TestAWARE, enable the examination of dynamic behaviours and reduce the efforts in the testing of mobile context-aware applications, as discussed in similar work [22]. With such a dataset, testers can easily conduct regression testing in the future if they change the targeted application. Similarly, testers may reuse datasets to test other applications which process the same types of data as the targeted application.

When synchronising these three kinds of data in data replay, designers of testing tools should use:

1. the timestamps of sensory data and events. Typically, context management middleware (e.g., AWARE [10]) imprints a timestamp on each entry of sensory data and events. Designers should notice the format of timestamp, for example, in the unit of milliseconds or nanoseconds.
2. The frame numbers of an audio file. Audio files do not contain timestamps, but their sample rates are stable over time. For example, an audio file with 44.1kHz sample rate has 44100 samples per second. For audio files in a single channel (smartphone/tablet microphones normally generate audio in one channel), one frame contains one sample. Hence, in line with Algorithm 1, designers can use frame numbers to synchronise audio and other two types of data. To improve replay efficiency, designers can apply a buffer to replay a number of samples per time.

6.4.2 Multiple Sources of Testing Data. To encourage the reuse of datasets for faster and cheaper testing, it is important for testing tools to support multiple sources of testing data. However, none of existing tools allow testers to import a dataset from different sources. This issue lowers the chance of dataset reuse, possibly causing more labour, time and cost in testing. For example, rather than downloading an existing dataset from the Internet, testers must record a new dataset for MobiPlay [22] to replay in testing. Comparatively, using TestAWARE, testers can import data from online databases, local device storage (e.g., data files in AWARE format) and the manipulator (e.g., creating a dataset with manipulated values). Furthermore, testers can also lean on the record-and-replay testing method because TestAWARE uses the data format of AWARE. That is, testers can first record data using AWARE and then replay the recorded data using TestAWARE.

To support multiple data sources, developers of testing tools may follow the same implementation of TestAWARE. Data from online databases can be downloaded by common query statements such as SQL. Unlike directly downloading a file, this method has sufficient safety under data encryption and access control. Access to local device storage (e.g., record-and-replay) is a trivial task for mobile developers. To implement a data manipulator, developers can refer to the data format of the testing tool. To improve dataset reuse, they should use the data format of a popular context management middleware that is already used by previous projects and studies, rather than inventing a new format. Moreover, if they refer to an open-source context management middleware, they can save significant time by reusing its code of data management.

6.4.3 Black-box Testing. Existing testing tools mostly emphasise the support of black-box testing. However, when using some of them, testers must modify the way the targeted application receives data, such as reading data from files on MobiPlay [22]. TestAWARE also implements this feature using a mobile client running together with the targeted application. Using TestAWARE, testers do not have to change the targeted application (except applications based on audio data, because they listen to microphone. For audio replay without loss of audio quality, these applications should receive audio data from another channel.), because TestAWARE sends the data using Android Intent with the same data format as AWARE. To conduct black-box testing on applications, testers can efficiently switch off AWARE data collection (i.e., the normal way applications collect data) and start data replay of TestAWARE. Applications, ignorantly, receive the replayed data from the same channel. Hence, TestAWARE does not require coding skills from testers for black-box testing. A drawback of TestAWARE, as well as other tools, in black-box testing is that it only replays data for applications to process and does not explicitly report or locate bugs. Testers themselves should draw conclusions. If testers are sceptical about the application, they should further conduct white-box testing.

To enable black-box testing, developers of testing tools may adapt the mobile client part of TestAWARE. Importantly, to encourage testing conducted by testers without programming skills, developers should not modify the original way the targeted application collects data, unless there are technical difficulties (e.g., applications based on audio data must use another channel to receive replayed data, rather than listening to microphone). Regarding the help in bug detection, black-box testing can hardly explicitly report or locate bugs because of the ignorance of internal details. Developers may implement CPU and memory monitoring features which can provide indirect evidence for testers to infer the existence of some flaws such as memory leakage.

6.4.4 White-box Testing. For general mobile context-aware applications, none of extant tools support white-box testing. Using TestAWARE, testers can conduct white-box testing by its APIs of the code library provided in a Java's JAR package. To check the internal details of applications, testers can output values or apply assertions (i.e., comparing the actual value and the expected value) on values inside the source code. Using the code library of TestAWARE, testers can record the dynamic behaviours of applications at run time which lead to accurate bug localisation and careful examination from low levels to high levels, as demonstrated in our user study. Furthermore, the code library allows testers to automate dataset management (i.e., downloading data, replaying data and stopping replay), indicating that testers can perform large-scale testing on a large number of devices or emulators. To test applications based on audio data, testers must change the way applications receive data due to the speciality of microphone data stream. Based on the audio data receiving API in the code library, testers are able to include audio as a data source in the data replay.

As highlighted by our work, white-box testing is a promising and useful feature which developers of testing tools may implement in future work. They can employ the mechanism of the code library, allowing

testers to insert testing code inside their applications. To support the examination of applications' internal behaviours, developers should design the commands and a result recorder for value output and assertions. For testing automation, developers should create commands for the controls of data management and replay. For the cases where testers have to change the way applications receive data, it is necessary for developers to provide corresponding data receiving commands for these applications to collect data from a new channel. Beyond TestAWARE, developers may include code coverage criteria, such as statement coverage and decision coverage, in their tools to assess the completeness of white-box testing.

6.4.5 Non-functional Testing. A small number of extant tools are specialised for testing non-functional properties of mobile applications, including machine learning performance, processing speed and power consumption estimation. However, they do not provide any features for functional testing at the same time, causing testers to spend more time and efforts on a complete testing process using different tools. This issue raises considerable challenges for regression testing which happens after every new change in software. To avoid this problem, testers can use TestAWARE which is able to conduct non-functional testing in parallel with functional testing. For the evaluation of machine learning performance and processing speed, testers can insert corresponding commands which record machine learning results and execution timestamp at the run time of programs. For the estimation of power consumption, testers need to provide a sensor power model for a specific device because each sensor on different devices varies in power use. TestAWARE records all the results on the device storage and reports them via the UI of the mobile client. Testers can also analyse each entry of the results by reading the data from the device storage.

Developers of testing tools should provide similar support for non-functional testing. For machine learning performance, they may implement more measures (e.g., Receiver Operator Characteristic (ROC) Area) beyond precision and recall. To achieve this, developers need to collect more run-time information from different machine learning algorithms. When recording timestamps for the measurement of processing speed, TestAWARE only allows testers to provide a name of the test. To support better evaluation of processing speed, developers can apply an assertion feature, which can be used by testers to compare the actual and the expected execution time. With this feature, testers can better find the bottlenecks in the programs in terms of processing speed. For the measurement of power consumption, developers can adapt the estimation method used by TestAWARE. Furthermore, they can attempt to monitor the actual power use of the targeted application on device battery. However, this is difficult because the operating system and testing tool also consume power at the run time of the targeted application. Developers could investigate how to accurately distinguish the power consumption from different sources.

6.4.6 Testing on Physical Devices and Emulators. To examine properties such as compatibility and processing speed, it is necessary for testers to run their applications on physical devices. Using TestAWARE, testers can run tests for their applications running on any smartphone and tablet with Android OS, which is the most popular mobile platform in the market. Also, supporting physical devices enables the data fusion of replayed data and the data collected by sensors in real time. For example, to test location-based applications, testers can use real-time GPS coordinates (i.e., using the GPS location from AWARE middleware) together with replayed data for other sensors (i.e., using TestAWARE data replayer). In some cases (e.g., physical devices do not have an OS version required in testing), testers need emulators to conduct testing. With TestAWARE, testers can also perform data replay and functional/non-functional testing on emulators. Regarding non-functional testing on emulators, the measurement of processing speed depends on the PC hardware of emulators, providing limited usefulness. Based on the results of our experiments on the maximal replay speed of data replay, testers should be aware that PC-based emulators can replay data significantly faster than actual smartphones and tablets. Testers can take advantage of PC-based emulators in testing scenarios where testers aim to efficiently replay longitudinal datasets in short testing time.

To enable the support of physical devices and emulators, developers of testing tools can simply implement their tools as applications on the targeted OS platform. For Android, developers can reuse the implementation of TestAWARE. As Android is implemented in Java, developers can make a Java's JAR package as a code library for the API commands of white-box testing. For other platforms, developers should use the corresponding programming languages for the implementation of testing tools and code libraries (e.g., Swift

for iOS, C# for Windows Phone). Note that the mechanism of data replay functions may be restricted by different mobile platforms (e.g., the sandbox policy of iOS higher than 7.0 bans inter-process communication).

6.4.7 Benefits of using TestAWARE. In summary of the above aspects, we identify a number of unique features of TestAWARE that can benefit testers in testing mobile context-aware applications, beyond extant testing tools. In Table 4 we demonstrate the benefits brought by each unique feature of TestAWARE.

With TestAWARE, testers can conduct new types of testing: testing applications collecting audio data, and white-box testing. When initialising synchronised replay with any of sensory, event and audio data, testers can define the speed of data replay for tests which are faster or slower than the real-time clock. To alleviate the difficulties of finding testing data, TestAWARE improves the chance of dataset reuse. To reduce the time and cost in testing, TestAWARE automates and simplifies the data preparation phase, the process of black-box testing and non-functional testing. Also, to save time in white-box testing and non-functional testing, TestAWARE can conduct these two at the same time by providing a code library with API commands.

Table 4. Benefits of using TestAWARE.

Feature	Benefit
Supporting synchronised replay of sensory, event and audio data, with replay speed setting	Enabling the testing of applications collecting any of sensory, event and audio data. Allowing tests which are faster or slower than the real-time clock.
Supporting testing data from online, local source and manipulation	Increasing the chance of dataset reuse. Simplifying the data preparation phase before tests.
Supporting black-box testing without requiring any modification for receiving data	Simplifying the process of black-box testing. Encouraging tests performed by testers which do not have corresponding programming skills.
Enabling white-box testing	Allowing testers to examine internal details of application and locate bugs from low levels to high levels.
Providing a code library for white-box testing and non-functional testing	Automating data preparation, white-box testing and non-functional testing. Simplifying non-functional testing. Allowing testers to conduct white-box testing and non-functional testing at the same time.

6.5 Limitations and Future Work

TestAWARE is implemented on Android. We are aware that the fragmentation of Android causes the inconsistency of sensor value representations from different hardware manufacturers. E.g., for the proximity sensor, some manufacturers use {0, 1} to represent a negative or positive. However, other manufacturers use {0, 15} or {0, 100}. These representations are all valid on Android platforms. If sensor values are collected on one device, it may be challenging for another device to recognise the values of the data replay in the testing. Developers may have to transform the representations of the datasets for testing their applications.

Regarding the implementation of TestAWARE on iOS, we found that the sandbox policy of iOS higher than 7.0 does not allow any inter-process communication, meaning that the targeted application can never receive the replayed data from an external testing tool. To perform effective data replaying, a tool must act as a module inside the targeted application (e.g., as a combination of a file reader and data replayer which the targeted application can import). However, this implies the modification of the source code of the targeted application, causing black-box testing to become impossible. Future research can investigate the suitable design of the tool for iOS, as well as other unpopular mobile platforms.

In our experiments, we quantified the maximal replay speed of audio, sensor and event data on smartphones, tablets and PC-based emulators. The experiment used one thread and one type of data per time.

In practice, the testing often involves heterogeneous data in the replay. When replaying multiple data sources at a high speed, the maximal replay speed may reduce because the number of CPU cores may be lower than the number of data sources.

For future work, we plan to include video data in the replay since most smartphones and tablets have dual cameras. Also, smartwatches with diverse sensors are becoming increasingly popular recently. We plan to design a smartwatch version of TestAWARE for the testing of smartwatch-based context-aware applications.

7 CONCLUSION

In this paper, we present TestAWARE, a laboratory-oriented testing tool for mobile context-aware applications, which can download, replay and construct contextual data on either physical devices or emulators. To support both black-box and white-box testing, TestAWARE is designed and implemented as a novel architecture with two components: a mobile client and code library. In black-box testing, developers can manage data replay from the mobile client without writing testing scripts or changing the source code of the targeted application. In white-box testing, the code library supports developers to automate data replay by writing testing scripts and conduct functional examinations using run-time assertions. With the code library, developers can also test non-functional properties of the targeted application, including machine learning performance, energy use and processing speed in real-time clock. We evaluated TestAWARE by quantifying its maximal data replay speed and conducting a user study with 13 professional developers. We found that PC-based emulators can replay data significantly faster than physical devices including smartphones and tablets. The results of the user study confirm the usefulness of TestAWARE in the testing of mobile context-aware applications in the laboratory settings. In the scope of testing mobile context-aware applications, our work provides multiple implications to conduct tests and testing tool design, which can guide both testers and developers (including researchers) of testing tools.

ACKNOWLEDGMENTS

This work is partially funded by SocialNUI, the Academy of Finland (Grants 276786-AWARE, 286386-CPDSS, 285459-iSCIENCE, 304925-CARE), the European Commission (Grants PCIG11-GA-2012-322138, 6AIKA-A71143-AKAI), Marie Skłodowska-Curie Actions (645706-GRAGE) and University of Oulu (Grants ITEE-2016-SA-13, and ITEE-2016-SA-20).

REFERENCES

[1] Domenico Amalfitano, Anna R. Fasolino and Porfirio Tramontana. 2011. A GUI Crawling-Based Technique for Android Mobile Application Testing. In *International Conference on Software Testing, Verification and Validation Workshops*, IEEE, 252-261. http://dx.doi.org/10.1109/ICSTW.2011.77.

[2] Domenico Amalfitano, Anna R. Fasolino, Porfirio Tramontana and Nicola Amatucci. 2013. Considering Context Events in Event-Based Testing of Mobile Applications. In *International Conference on Software Testing, Verification and Validation Workshops*, IEEE, 126-133. http://dx.doi.org/10.1109/ICSTW.2013.22.

[3] Application Exerciser Monkey. Retrieved 17/12/2014 from http://developer.android.com/tools/help/monkey.html

[4] Martin Atzmueller and Katy Hilgenberg. 2013. Towards Capturing Social Interactions with SDCF: An Extensible Framework for Mobile Sensing and Ubiquitous Data Collection. In *Proceedings of the 4th International Workshop on Modeling Social Media*, ACM, 6:1-6:4. http://dx.doi.org/10.1145/2463656.2463662.

[5] Rajesh K. Balan, Darren Gergle, Mahadev Satyanarayanan and James Herbsleb. 2007. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, 272-285.

[6] Szymon Bobek, Sebastian Dziadzio, Paweł Jaciów, Mateusz Ślażyński and Grzegorz J. Nalepa. 2015. *Understanding Context with ContextViewer – Tool for Visualization and Initial Preprocessing of Mobile Sensors Data.* Springer International Publishing.

[7] David Chu, Nicholas D. Lane, Ted T. Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li and Feng Zhao. 2011. Balancing Energy, Latency and Accuracy for Mobile Sensor Data Classification. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ACM, 54-67. http://dx.doi.org/10.1145/2070942.2070949.

[8] Karen Church and Barry Smyth. 2008. Understanding the intent behind mobile information needs. In *Proceedingsc of the 13th international conference on Intelligent user interfaces - IUI '09*, 247-256. http://dx.doi.org/10.1145/1502650.1502686.

[9] Context Simulator (KnowMe). Retrieved 14/04/2016 from http://glados.kis.agh.edu.pl/doku.php?id=pub:software:contextsimulator:start

[10] Denzil Ferreira, Vassilis Kostakos and Anind K. Dey. 2015. AWARE: mobile context instrumentation framework. *Frontiers in ICT* 2, 6: 1-9. http://dx.doi.org/10.3389/fict.2015.00006.
[11] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim and Todd Millstein. 2013. RERAN: Timing- and touch-sensitive record and replay for Android. In *International Conference on Software Engineering*, IEEE, 72-81. http://dx.doi.org/10.1109/ICSE.2013.6606553.
[12] Tobias Griebe and Volker Gruhn. 2014. A model-based approach to test automation for context-aware mobile applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 420-427.
[13] Matthew Halpern, Yuhao Zhu, Ramesh Peri and Vijay J. Reddi. 2015. Mosaic: cross-platform user-interaction record and replay for the fragmented android ecosystem. In *International Symposium on Performance Analysis of Systems and Software*, IEEE, 215-224. http://dx.doi.org/10.1109/ISPASS.2015.7095807.
[14] Kaasila, Denzil Ferreira, Vassilis Kostakos and Timo Ojala. 2012. Testdroid: automated remote UI testing on Android. In *International Conference on Mobile and Ubiquitous Multimedia*, ACM, 28:1-28:4. http://dx.doi.org/10.1145/2406367.2406402.
[15] Reed Larson and Mihaly Csikszentmihalyi. 1983. The Experience Sampling Method. In *Flow and the Foundations of Positive Psychology* (eds.). Wiley Jossey-Bass, San Francisco, 15, 41-56.
[16] Heng Lu, W. K. Chan and T. H. Tse. 2006. Testing Context-aware Middleware-centric Programs: A Data Flow Approach and an RFID-based Experimentation. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ACM, 242-252. http://dx.doi.org/10.1145/1181775.1181805.
[17] Chu Luo, Miikka Kuutila, Simon Klakegg, Denzil Ferreira, Huber Flores, Jorge Goncalves, Vassilis Kostakos and Mika Mäntylä. 2016. How to Validate Mobile Crowdsourcing Design? Leveraging Data Integration in Prototype Testing. In *International Joint Conference on Pervasive and Ubiquitous Computing Adjunct*, ACM. http://dx.doi.org/10.1145/2968219.2968586.
[18] Qi Luo, Denys Poshyvanyk, Aswathy Nair and Mark Grechanik. 2016. FOREPOST: A Tool for Detecting Performance Problems with Feedback-driven Learning Software Testing. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ACM, 593-596. http://dx.doi.org/10.1145/2889160.2889164.
[19] Chulhong Min, Seungchul Lee, Changhun Lee, Youngki Lee, Seungwoo Kang, Seungpyo Choi, Wonjung Kim and Junehwa Song. 2016. PADA: Power-aware Development Assistant for Mobile Sensing Applications. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ACM, 946-957. http://dx.doi.org/10.1145/2971648.2971676.
[20] monkeyrunner | Android Developers. Retrieved 03/05/2016 from http://developer.android.com/tools/help/monkeyrunner_concepts.html
[21] Henry Muccini, Antonio D. Francesco and Patrizio Esposito. 2012. Software testing of mobile applications: Challenges and future research directions. In *Automation of Software Test (AST), 2012 7th International Workshop on*, 29-35.
[22] Zhengrui Qin, Yutao Tang, Ed Novak and Qun Li. 2016. MobiPlay: A Remote Execution Based Record-and-replay Tool for Mobile Applications. In *Proceedings of the 38th International Conference on Software Engineering*, ACM, 571-582. http://dx.doi.org/10.1145/2884781.2884854.
[23] Kiran K. Rachuri, Mirco Musolesi, Cecilia Mascolo, Peter J. Rentfrow, Chris Longworth and Andrius Aucinas. 2010. EmotionSense: a mobile phones based adaptive platform for experimental social psychology research. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, ACM, 281-290. http://dx.doi.org/10.1145/1864349.1864393.
[24] Luis Roalter, Andreas Moller, Stefan Diewald and Matthias Kranz. 2011. Developing intelligent environments: A development tool chain for creation, testing and simulation of smart and intelligent environments. In *Intelligent Environments (IE), 2011 7th International Conference on*, 214-221.
[25] M Satyanarayanan. 2001. Pervasive computing: Vision and challenges. *Personal Communications, IEEE* 8, 4: 10-17. http://dx.doi.org/10.1109/98.943998.
[26] Choonsung Shin, Jin-Hyuk Hong and Anind K. Dey. 2012. Understanding and Prediction of Mobile Application Usage for Smart Phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ACM, 173-182. http://dx.doi.org/10.1145/2370216.2370243.
[27] Melanie Swan. 2013. The Quantified Self: Fundamental Disruption in Big Data Science and Biological Discovery. *Big Data* 1, 2: 85-99. http://dx.doi.org/10.1089/big.2012.0002.
[28] Ralf Tonjes, Eike S. Reetz, Marten Fischer and Daniel Kuemper. 2015. Automated Testing of Context-Aware Applications. In *Vehicular Technology Conference*, IEEE, 1-5. http://dx.doi.org/10.1109/VTCFall.2015.7390847.
[29] Niels van Berkel, Chu Luo, Theodoros Anagnostopoulos, Denzil Ferreira, Jorge Goncalves, Simo Hosio and Vassilis Kostakos. 2016. A Systematic Assessment of Smartphone Usage Gaps. In *Conference on Human Factors in Computing Systems*, ACM, 4711-4721. http://dx.doi.org/10.1145/2858036.2858348.
[30] Martin White, Mario Linares-Vásquez, Peter Johnson, Carlos Bernal-Cárdenas and Denys Poshyvanyk. 2015. Generating Reproducible and Replayable Bug Reports from Android Application Crashes. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, IEEE Press, 48-59.

Received November 2016; revised May 2017; accepted July 2017

Chapter 4

Validating real-time sensing performance

4.1 Aims and hypotheses

This chapter explores how the real-time sensing performance of context-driven mobile applications can be efficiently and systematically validated in the laboratory. We proposed a validation approach with the following aims:

1. **Quantifying the distortion of modified sensor signals.** Mobile applications may modify sensor signals in real-time sensing (e.g., to compress the sensing data [83], or to protect sensitive data by hiding it into sensor signals). Our approach aims to quantify the amount of error caused by modification in sensor signals.

2. **Testing different sensing frequencies and device models.** Our approach aims to investigate how performance varies across different sensing frequencies and device models.

3. **Testing different sensor types.** Our approach attempts to explore how different sensor types perform under identical settings.

4. **Measuring processing speed.** Our approach aims to measure the processing speed of real-time sensing in two aspects: execution time and payload count (i.e., how many sensor readings can be processed in a given period of time).

5. **Measuring CPU utilisation.** Our approach aims to investigate how CPU utilisation of certain executions can be measured in real-time sensing.

6. **Enabling design phase testing in the laboratory.** Our approach aims to examine these performance properties of real-time sensing at the design phase, without implemented software or with the minimal efforts of implementing a software prototype. Also, our approach intends for testers to launch all tests in the laboratory.

We hypothesised that our approach can achieve efficient and systematic laboratory-based validation for the real-time sensing performance of context-driven mobile applications. With an exemplar targeted application that uses a data hiding method to encrypt and embed sensitive information into streams of real-time sensor data, our approach is detailed in the attached publication in Section 4.2.

4.2 Publication

Chu Luo, Angelos Fylakis, Juha Partala, Simon Klakegg, Jorge Goncalves, Kaitai Liang, Tapio Seppänen, and Vassilis Kostakos. A data hiding approach for sensitive smartphone data. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 557-568. ACM, 2016. `https://doi.org/10.1145/2971648.2971686`.

A Data Hiding Approach for Sensitive Smartphone Data

Chu Luo[1], Angelos Fylakis[2], Juha Partala[2], Simon Klakegg[1], Jorge Goncalves[1], Kaitai Liang[3], Tapio Seppänen[2], Vassilis Kostakos[1]
[1]Center for Ubiquitous Computing, University of Oulu, Finland
[2]Center for Machine Vision and Signal Analysis, University of Oulu, Finland
[3]Department of Computer Science, Aalto University, Finland
[1,2]firstname.lastname@ee.oulu.fi, [3]kaitai.liang@aalto.fi

ABSTRACT
We develop and evaluate a data hiding method that enables smartphones to encrypt and embed sensitive information into carrier streams of sensor data. Our evaluation considers multiple handsets and a variety of data types, and we demonstrate that our method has a computational cost that allows real-time data hiding on smartphones with negligible distortion of the carrier stream. These characteristics make it suitable for smartphone applications involving privacy-sensitive data such as medical monitoring systems and digital forensics tools.

Author Keywords
Smartphones; ubiquitous computing; privacy protections; digital signal processing; mobile and wireless security.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
We present and evaluate a data hiding method for smartphone sensing, which enables sensing applications to encrypt and embed sensitive data or identification codes within other data streams of a smartphone. Our method is motivated by the increasing diversity of sensor data that mobile devices generate, and the growing ecosystems of services that store, process, and share this data.

The constellation of personal devices that we regularly use now includes smartphones, smartwatches, tablets, fitness sensors, and a variety of domestic appliances. These devices contain a growing set of increasingly sophisticated sensors which improve interaction and provide new services. The richness and volume of this data have given rise to research opportunities for the UbiComp community and beyond. For instance, research often demonstrates how smartphone data that we previously discarded as noise can actually contain valuable information [1]. Furthermore, research on quantified self, self-monitoring, and e-health aims to harness the data that our devices generate.

This trend has motivated scientists across academia and industry to build platforms that collect, analyse, visualise, and share increasing amounts of end-user data generated from personal devices. It has become a norm for studies to instrument personal devices of volunteer participants (recruited both in-person or via app-stores), and much of this data may eventually become available for other scientists or the public in general. Initiatives such as Crawdad, Crowdsignals, and Nokia's Lausanne Data Collection Campaign are examples of how smartphone "traces" may be shared after the completion of an experiment. Similarly, the quantified-self movement has given rise to a large number of platforms where users may upload their health-related sensor data. While certain platforms may be free or come with associated costs, users typically share their data in exchange for a service.

Often, multi-stream data from a user's device is treated as a coherent data unit. For instance, a typical experiment may simultaneously collect accelerometer, heart rate, and GPS data. This set of sensor streams may then be uploaded to a server for analysis, visualisation and sharing. This approach to "bundling" sensor streams has two important downsides. Firstly, it treats all sensor streams unilaterally, overlooking the unique privacy aspects of each individual sensor stream. For instance, accelerometer data may not be as sensitive as GPS data. Secondly, users practically relent control of their data once it leaves their devices.

Our work proposes a data hiding approach to address the privacy needs that arise when users share smartphone sensor data with scientists and platforms. Crucially, our technique is transparent, meaning that it is compatible with existing platforms and tools. As a result, users can maintain control of their sensitive data even after sharing it through online platforms, without having to give up those services. Additionally, the technique allows us to verify the integrity of embedded sensitive data, for example to confirm that no tampering has taken place in the form of record removal, decimal rounding, or filtering. Finally, our technique allows users to prove ownership of sensor or healthcare data that they have shared, and as such provides spoof resistance against tampered medical sensor data [31].

UbiComp '16, September 12 - 16, 2016, Heidelberg, Germany

ACM 978-1-4503-4461-6/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2971648.2971686

BACKGROUND

Data hiding is an application domain of digital watermarking techniques [7]. Traditionally, watermarks are found in official documents, and carry information about the object in which they are found. Watermarks are designed in way so that they are difficult to reproduce or counterfeit [12]. For example, banknotes have watermarks in the form of figures that become visible only under certain conditions.

The practice of watermarking can be defined as imperceptibly altering an object to embed a message about it [12]. Embedding a watermark w into a host object C produces a new object C_w, such that w can be reliably located and extracted even after C_w has been subjected to transformations [8].

In digital watermarking the host object C is a carrier signal of information, and the watermark w is a digital marker. The watermarking process is achieved through the introduction of errors not detectable by human perception [11]. Similar to traditional watermarking, digital watermarks can only be perceptible under specific conditions such as after using special extracting algorithms [33]. Unlike traditional watermarking, in digital watermarking when the carrier signal is copied or transferred, the watermark is also carried with the copy.

Watermarking methods have been used in various applications including digital audio, images or videos. Typically, these are used for owner identification, content authentication and copy control [12]. Similarly, data hiding is particularly popular with biomedical data because of the need to imperceptibly carry metadata or additional sensitive data such as name, ID, or sensitive medical data [37]. In biomedical research, data hiding techniques embed data reversibly, since it is important to use it in its original state in the analyses. In our case though, we can be flexible as the host data is not sensitive. In this context, data hiding can improve management efficiency, provides an additional layer of security, and can ensure confidentiality, availability and reliability [7]. As such, data hiding allows us to embed a set of metadata or sensitive data, imperceptibly, within another data set or digital file.

Finally, we observe that cryptographic techniques offer orthogonal benefits regarding these concerns. For this reason, data hiding techniques often encrypt the data before it is hidden. However, the key advantage of data hiding is the "physical" binding between carrier signal and digital marker in a manner that is transparent to computational infrastructure, can survive data migration, and does not give rise to software compatibility issues.

Properties of Data Hiding Techniques

Multiple data hiding techniques exist, and they can be classified in terms of three key properties [18].

- Robustness: Robust techniques are those where data can be extracted successfully even after the carrier signal has undergone malicious attacks, modifications or transformations. This feature is particularly desirable in cases where the host is prone to modifications, either intentional, or unintentional. An example would be lossy compression. Fragile techniques are those where minor distortions affect the hidden data. This can serve useful tamper-proofing purposes (e.g., loss of hidden data can reveal and localise modification of data).
- Imperceptibility: Data hiding techniques are considered as imperceptible when data is imperceptible to human under typical use. Hidden data can only be extracted algorithmically by an authorized user. Good imperceptibility also suggests high fidelity between the original work and the one containing data.
- Capacity: Refers to the size of the payload that can be encoded within a unit of a host object.

Traditionally in data hiding literature, there is a tradeoff between these three properties. Depending on the application domain, the priority of these properties varies. An additional constraint in the case of smartphones and mobile sensors is energy consumption and computational complexity. Previous work has highlighted that data hiding is more efficient than cryptography in terms of complexity and energy usage, and therefore more appropriate for resource-constrained hardware [18].

Data Hiding and Smartphone Sensor Data

Smartphone sensing techniques introduce a variety of applications and opportunities, such as activity recognition, health monitoring and intelligent transportation. However, smartphone sensor data may contain sensitive information, including GPS location, medical states (e.g., heart rate and travelled steps) and user profiles (e.g., identity, age, gender and calendar reminders). This challenges researchers in the design of smartphone sensing systems [20].

Although researchers can alleviate these problems using cryptography and privacy-preserving data mining techniques, existing approaches are still insufficient to keep information imperceptible or to prove the authenticity of sensor data [19, 22]. For this reason, data hiding techniques can benefit users, for example by hiding sensitive data within non-sensitive data. Furthermore, data hiding techniques can be used to prove ownership of smartphone sensor data without compromising anonymity. By embedding identification information into sensor data, sensing systems that collect data from multiple users (e.g. in crowdsensing) can easily verify the source of data and filter untrusted sources without establishing secure access APIs or explicit authentication mechanisms.

Many projects have considered data hiding techniques, especially watermark-based methods, in smartphone-driven scenarios. Miao et al. [27] developed an Android application that uses digital watermarks to protect the ownership and integrity of digital photographs. They showed that the proposed approach can resist some

common attacks, such as contrast change and compression. Zhou et al. [40] developed a system named AppInk that generates watermarked apps from the source code of original apps, to detect unauthorised apps which are repackaged by attackers. Suzuki et al. [35] developed a video annotation system which embeds real-time high-frequency audio watermarks into video data of a smartphone camera. Because high-frequency audio is inaudible to humans, the audio quality of watermarked video data is not compromised. Hence, users can add annotations into video in the form of audio watermarks.

Furthermore, data hiding has been used as a barcode-like mechanism. For example, previous work embeds hyperlinks within posters or videos [9], such that a mobile device can decode this information but it is not perceptible to humans. Similarly, researchers have shown how to embed information within an audio channel transmitted over loudspeakers [26] or the phone [30], a technique that can be used for ad-hoc secure pairing, verification, and synchronisation.

In the context of sensor networks that may have to operate in untrusted environments, data hiding can meet the requirements of data integrity and authentication in communication. For example, Wang et al. [38] proposed an adaptive watermarking approach to achieve secure image transmission with low distortion and energy cost. Similarly, Zhang et al. [39] presented an end-to-end authentication scheme that employs watermarking for secure data aggregation. In these cases, watermarks are embedded by each sensor node, and the server can verify the sources of the incoming data despite an untrusted communication network.

Our work builds on previous research in many ways. The recent proliferation of scientific and commercial platforms for sensor data has given rise to the need to consider "sensing" as the application itself. Therefore, we aim to provide users a transparent way to embed one set of smartphone sensor data within another. This will allow users to adopt services on a variety of platforms without necessarily trusting them with their sensitive data. Much like sensor nodes operating in an untrusted network [38, 39], we can enable users' personal devices to share sensor data with each other via untrusted platforms. Additionally, our approach establishes a physical binding between a sensor stream and annotation data, either to prove ownership of the sensor data (as demonstrated with photos [27]), to provide additional context (as has been shown for videos [35]), or for ad-hoc communication purposes (as has been used in ad-hoc pairing [26, 30]).

STUDY

Our objective is to investigate the feasibility of hiding one sensor stream within another on a smartphone. Due to the plethora of sensors, it is important to identify their main types for our purposes. Modern smartphones can provide sensor data across the following broad categories [21]:

- hardware sensors that include motion sensors (e.g., accelerometer and gyroscope), position sensors (e.g., GPS and magnetometer), environmental sensors (e.g., light sensor and barometer), and multimedia hardware (e.g., microphone and dual-cameras).
- software sensors that include operating system data (e.g. CPU load, network connections, app usage) and application data (e.g. calendar data, browsing history, music listening data).
- human input, which captures phenomena that are imperceptible for hardware or software sensor, mainly using smartphone-based surveys and the Experience Sampling Method [23].

Given the diversity of data sources, there are two important characteristics that we consider for data hiding purposes:

- *frequency*: some sensor data may be collected at high frequency, such as accelerometer and magnetometer data. Other sensors may provide data with much lower frequency, such as heart rate sensors, GPS, or human input text. Furthermore, some data may be constant, such as user identifier, names and date of birth.
- *privacy sensitivity*: some data may be highly sensitive if exposed, such as date of birth, user identifier, or heart rate data. Other data may be less sensitive, for example accelerometer and gyroscope values.

In the diagram below we map out many of the possible smartphone sensors in terms of their frequency and sensitivity. Data hiding is ideal for hiding low-frequency sensitive data into high-frequency non-sensitive data. It is challenging to objectively map the privacy concerns that may be associated with any particular sensor, since they can vary across users and strongly depend on what other data is available. For instance, gyroscope is typically used together with accelerometer to detect activities such as walking, standing, sitting and lying can be recognised with high accuracy 96% [3], while on the other hand complex activities (e.g., cooking, cleaning and sweeping) are still considered challenging to recognise [13]. Therefore, we rely on subjective assessment, heuristics, and our review of literature [19, 22, 28] to rate the privacy concerns for each sensor. We summarise the different types of sensor data in terms of frequency and sensitivity in Figure 1.

DATA HIDING METHODOLOGY

In data hiding techniques the payload can be hidden either in the time or frequency domain of the carrier. We adopt a time domain technique, and specifically a substitutive insertion method: parts of the carrier signal are replaced by the payload signal. Specifically, we apply the Least Significant Bit (LSB) substitution scheme, replacing the carrier signal's least significant bits with bits of the payload. This approach enables embedding data of high rate and size, while causing relatively insignificant modifications to the original values [15, 29].

More importantly, our method is appropriate for real-time data hiding on limited-resource platforms such as smartphones, due to its low time complexity, and can therefore guarantee that the embedded data is synchronised with the carrier data. For example, let us assume that we need to embed a heart rate value into a stream of accelerometer values. We first obtain the binary representation of the heart rate value, which for instance can be a 32-bit integer. The next step is to embed each bit in the oncoming accelerometer stream, by replacing the LSBs of consecutive accelerometer values that have also been converted to binary form (see Figure 2). One bit is used for a flag to denote the existence of embedded data, and additional bits are used for the payload. Depending on how much we can afford to distort the carrier values, we can opt to replace two or more LSBs.

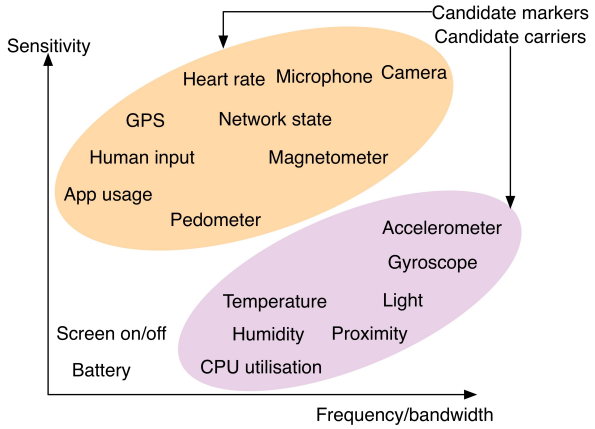


Figure 1. Sensitivity and frequency of different sensor data.

Authenticated encryption prior to embedding enables us to detect errors or tampering of the hidden data. Therefore, we apply the Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) [14, 32] to encrypt and authenticate the embedded data. GCM is a mode of operation that supports simultaneous encryption and authentication of a data stream in an efficient, parallelisable manner. To ensure the integrity of the data, an authentication tag is generated at the end of the input stream. The GCM mode is nonce-based which means that a unique public identifier called nonce or initialisation vector needs to be used for each set of data. Both the nonce and the secret encryption key are needed for decryption and to check the integrity of the data. Any tampering of the encrypted hidden data will be detected during the decrypting phase.

For our evaluation we implemented our method on Android smartphones. We developed a pair of applications that run simultaneously and communicate via Intent messages within Android. The sensing application collects the sensitive data to be hidden and passes it to the data hiding application. The latter encrypts the received data, hides it into the carrier signal, and potentially stores it or transmits it to a third party. The two-application architecture was chosen to allow flexibility in practical scenarios, and to conduct a realistic assessment of performance. To extract the hidden data, the receiving party needs access to the carrier signal (sorted by timestamp), knowledge of the data types, how many LSBs are used, the pre-shared nonce, the authentication tag and encryption key.

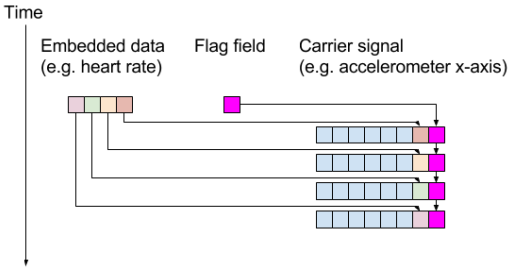


Figure 2. The Least Significant Bit (LSB) method replaces the least significant bits of the carrier signal with data to be embedded.

Distortion of the Carrier Signal

The distortion produced by hiding data into the carrier signal depends on the data type of the carrier signal and the number of LSBs. For example, given 32-bit integer types as the carrier signal, using n ($0<n<32$) LSBs for data hiding will produce an error from $-(2^n-1)$ to 2^n-1. The cases where floating-point numbers serve as carrier signals are more complex. Suppose the carrier signal is a 32-bit single precision floating number f [17] defined as

$$f = (-1)^s \times c \times 2^q, \qquad (1)$$

where s stands for the sign bit; c for the significand; and q for the exponent.

If we use n ($0<n<23$) LSBs (which determine the significand c) for data hiding, we must know the exponent value q (which depends on the magnitude of the floating number) to quantify the maximal amount of error $|E_{max}|$. It can be calculated as

$$|E_{max}| = \sum_{i=1}^{n} 2^{i-24+q} \qquad (2)$$

This formula reveals that a small number of LSBs will produce negligible errors with floating-point numbers. However, the absolute amount of error can be very high for large q. Although q can be up to 127 to represent a valid real number [17], smartphone sensors generally output much smaller readings in practice, such as accelerometer [25]. Thus, floating-point numbers are ideal carrier signals, since data hiding can have a negligible distortion on them.

EXPERIMENTAL DESIGN

We evaluated the performance of our approach by running experiments with off-the-shelf smartphones. Considering the availability of AES/GCM encryption, we selected 6 smartphones with Android OS version 5.0 or above:

Samsung Galaxy S6 edge (5.1.1); two samples of Motorola Moto G X1032 (5.1); LG Nexus 5 (5.1.1); Motorola Moto G2 (5.0.2); Yota YotaPhone 2 (5.0).

Experiment	Device	Marker	Carrier	LSBs
E1	All	Magnetometer (3 x 32-bit float) (normal, UI, game, fastest)	Accelerometer (3 x 32-bit float) (normal, UI, game, fastest)	2
E2	S6	Magnetometer (3 x 32-bit float) (normal, UI, game, fastest)	Accelerometer (3 x 32-bit float) (normal, UI, game, fastest)	3
E3	S6	Heart rate sensor (32-bit int) (normal, UI, game, fastest)	Accelerometer (1 x 32-bit float) (normal, UI, game, fastest)	2
E4	S6	GPS (3 x 64-bit double) (0.1, 0.2, 1, 10 Hz)	Accelerometer (3 x 32-bit float) (normal, UI, game, fastest)	2
E5	S6	Human input (8-bit char) (1, 10, 100, 200 Hz)	Accelerometer (1 x 32-bit float) (normal, UI, game, fastest)	2
E6	S6	Device ID (16 x 8-bit char)	Accelerometer (1 x 32-bit float) (normal, UI, game, fastest)	2

Table 1. Experimental parameters. The frequencies "normal, UI, game, fastest" are Android standards, and may perform differently across different handsets. The Least Significant Bit includes a 1-bit flag field.

We installed our pair of Android applications on each handset. We first launched the data hiding application to initialise the encryption and carrier signal. Then, we launched the sensing application to collect the data to be hidden, and pass it along for hiding. Because it is impractical to exhaust all the combinations of multiple variables, we designed 6 experiments to examine a range of conditions as summarised in Table 1.

We use the accelerometer as the carrier signal in all the experiments, and we consider 4 different sampling rates for it, as defined by Android. In E1 and E2 we hid magnetometer data at 4 different sampling rates. In E3 we hid heart rate data at 4 different sensing rates. In E4 we hid simulated streaming GPS data (64-bit double type in 3 dimensions: latitude, longitude and altitude) generated at 4 different frequencies. In E5, we hid simulated human input text at varying frequencies. In E6 we hid a device ID (an Android device ID has 16 characters).

For experiments E1, E2 and E4, we used 3 axes of the accelerometer as the carrier, since in those experiments we effectively had 3 streams of data to hide. In the other experiments only the *x* axis was used as a carrier. In experiments 5 and 6, the data hiding application used the 8-bit ASCII format. The experiments had a combination of sampling frequencies of the data to hide, and 4 frequencies of the carrier signal. Orthogonally, E1 had 6 different phones. Each condition ran for a period of 5 minutes, during which the performance of the system was monitored.

EXPERIMENTAL RESULTS

Figure 3 summarises the results in E1, where magnetometer data was encrypted and embedded into the accelerometer data using 4 different sampling rates on 6 handsets. The dark red shades represent the magnetometer records that were hidden, and the light blue shades above red shades represent the number of magnetometer records that could not be processed due to the too high bit rate of the payload, and therefore had to be dropped.

E1 primarily acted as a "stress test" to highlight performance differences across handsets. As such, we induced record dropping due to the relatively high volume of magnetometer data that we attempted to hide, as well as variances in the capabilities of the handsets. The results show that the sampling rate at "normal" and "UI" was consistent across handsets. However, the handsets performed substantially differently at the "Game" and "Fastest" sampling rates, for instance with the S6 outperforming G1 handsets by a factor of 2.

We further investigate the variation in the carrier frequency across handsets in E1. Figure 4 shows the average accelerometer delay, which denotes the time gap between two adjacent samples. Sensing delay is an indirect measure of the ability to execute data hiding.

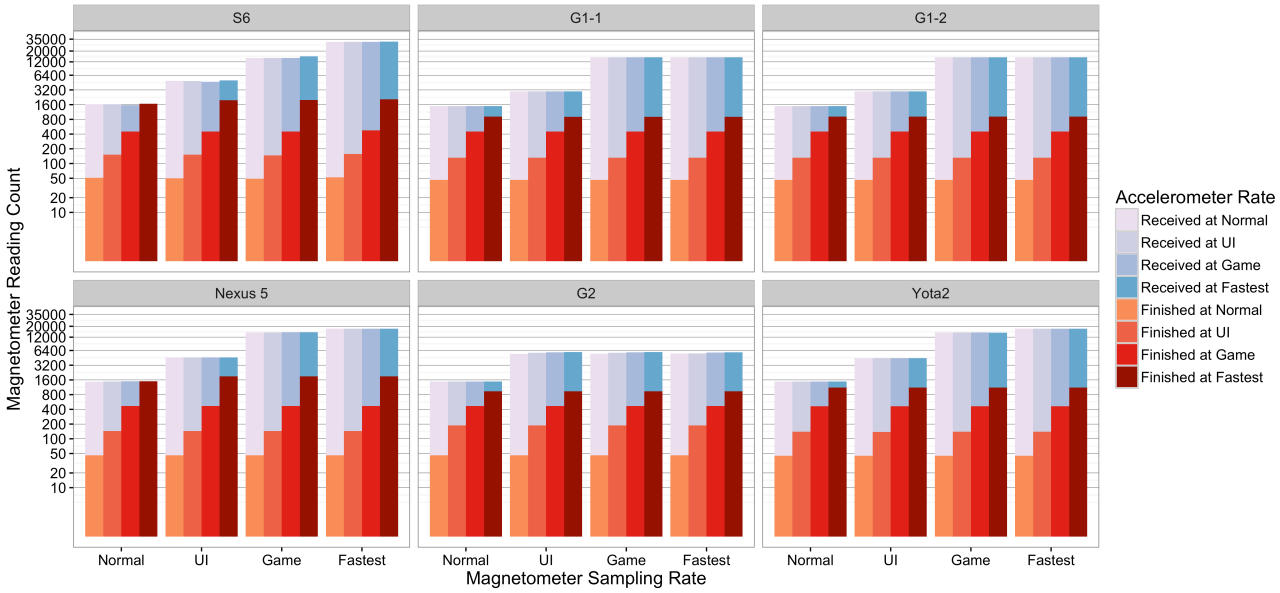


Figure 3. Results of E1. The number of magnetometer readings which are successfully hidden is shown in red, and those dropped is shown in blue. The *y* axis is on a base-2 logarithmic scale.

Based on the sensing delay, we can estimate the capacity of accelerometer as a carrier signal on each device. Figure 5 presents the capacity of one axis of the accelerometer for 2 LSBs (1 bit flag & 1 bit payload). We observe that at the fastest sampling rate, all handsets can provide a capacity of more than 10B/s, with the highest being 26.8B/s for the S6. If 3 axes are used, then the capacity increases by a factor of 3. In addition, the capacity increases proportionally for each additional payload bit we use. Therefore, we expect the S6 with 3 axes and 3 LSBs (1-bit flag & 2-bit payload) to provide $26.8 \times 3 \times 2 = 160$B/s capacity.

Once our data hiding application received a new magnetometer data reading, it executed AES/GCM encryption and hid the ciphertext bits into the incoming accelerometer records. When the ciphertext bits are more than the payload of one accelerometer record, phones have to embed the rest cipher bits into more incoming accelerometer records.

In Figure 6 we show the computational overhead that encryption induced in E1. Results show that, on average, all the handsets were able to finish the task of encryption plus data hiding for one sample within 0.6 ~ 6.2ms for any condition (max: 302.13ms due to CPU scheduling). Of this time, less than 0.2 ~ 0.8ms on average (max: 428.95) was spent on just data hiding.

Figure 7 shows the performance of the S6 handset across all experiments, and therefore for multiple data types. As expected, using an additional LSB in E2 doubled its capacity. In E3 we noted that the heart rate sensor hardware did not alter its sampling rate, contrary to Android API specifications. In E4, as expected, the results show that the number of GPS records we could hide was approximately half of the magnetometer in E1. In E5 the hidden data was simulated human entry text, which was on average 3 times faster than E1. In E6 we hid a Device Identification code, and therefore the sampling rate did not vary.

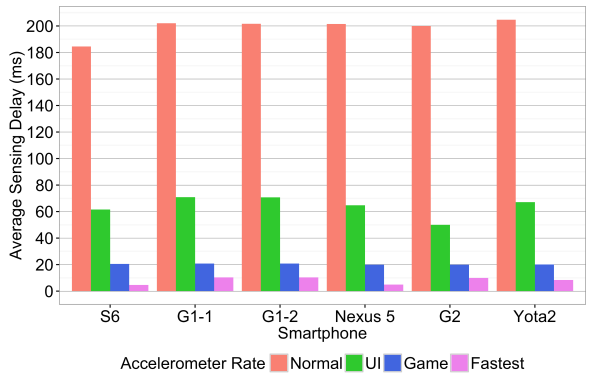


Figure 4. Average sensing delay of accelerometer for different handsets in E1.

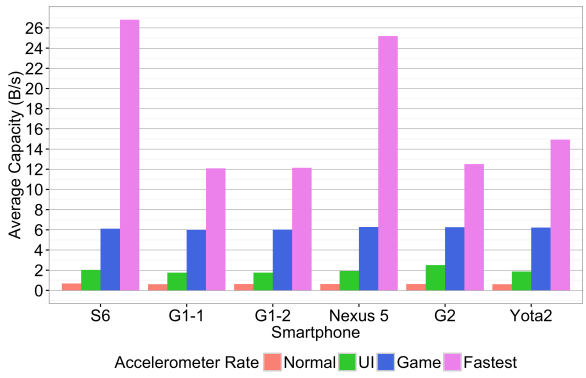


Figure 5. Average capacity using one-axis accelerometer carrier on 6 phones, using 2 LSBs (1 bit flag & 1 bit payload).

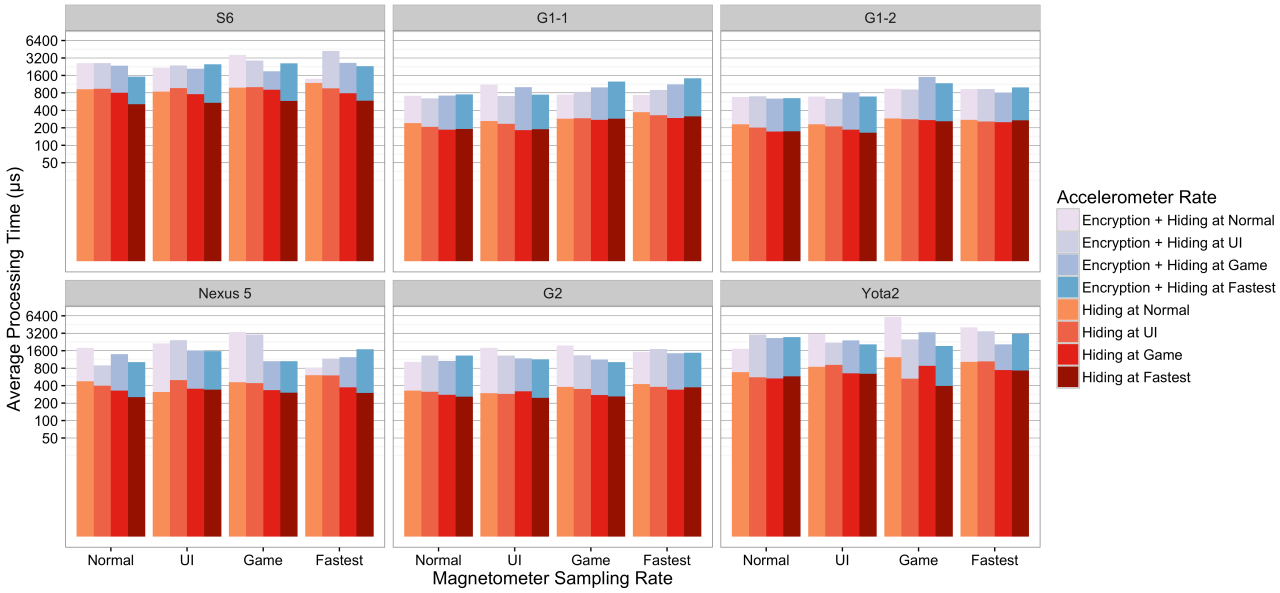


Figure 6. Average processing time for encryption & hiding (blue), or just hiding (red) in E1. This is the time needed for one magnetometer record. The *y* axis is in base-2 logarithmic scale.

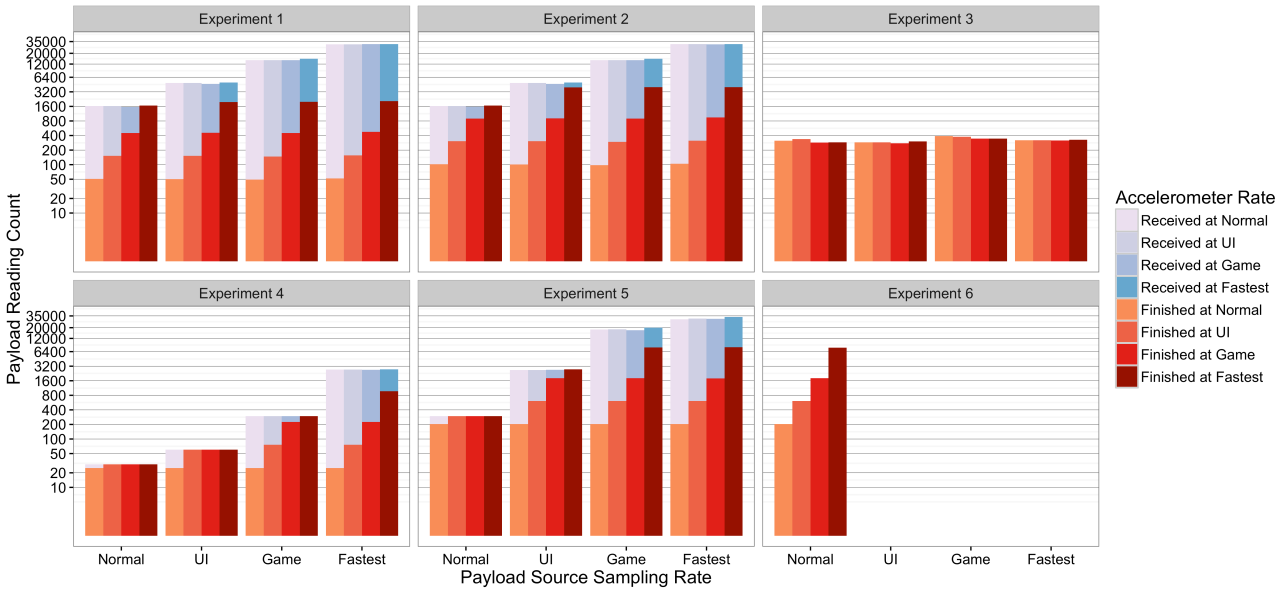


Figure 7. Performance of the S6 handset across all experiments. Number of readings which are successfully hidden is shown in red, and those dropped is shown in blue. The *y* axis is on a base-2 logarithmic scale.

Also, in E3 we observed that the accelerometer sampling rate was unexpectedly doubled compared to all other experiments (for UI speed: 30ms in E3 vs 60ms in other experiments). This is a phenomenon that we were able to reliably reproduce. Given the lack of official documentation we believe that on this particular handset, using the heart rate sensor triggers additional mechanisms that increase the sampling rate of the accelerometer. Figure 8 shows the average processing time of encryption and data hiding on S6 across all experiments. Considering encryption plus data hiding (blue), the average processing time follows the complexity of payload types and the number of LSBs: E1 (32-bit float on 3 axes, 2 LSBs): 2.49ms; E2 (32-bit float on 3 axes, 3 LSBs): 2.56ms; E3 (32-bit int, 2 LSBs): 1.14ms; E4 (64-bit double on 3 axes, 2 LSBs): 2.92ms; E5 (8-bit ASCII, 2 LSBs): 0.74ms; E6 (8-bit ASCII, 2 LSBs): 0.52ms. Similar to the worst case (among all handsets) in E1, the worst cases in E2-E6 ranged from 40.76ms to 380.67ms. When considering only data hiding (red), the S6 handset was able to finish within 0.9ms on average across all 6 experiments. The worst cases in E2-E6 ranged from 57.29ms to 593.36ms.

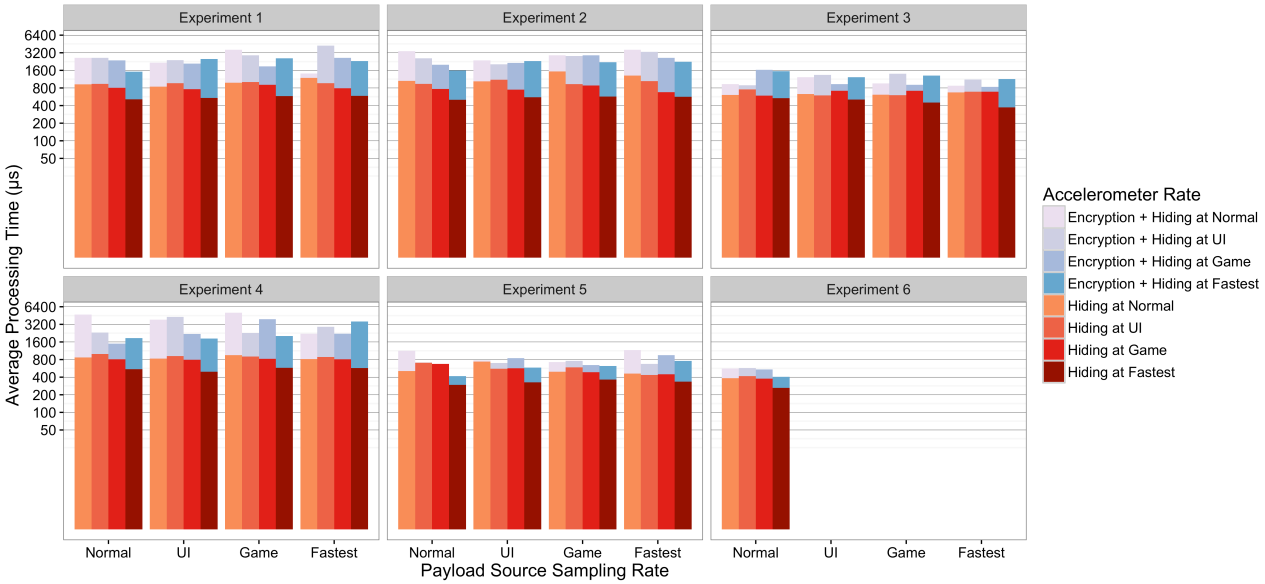


Figure 8. Average processing time (S6 handset across all experiments) for encryption & hiding (blue), or just hiding (red) in E1. The *y* axis is in base-2 logarithmic scale.

CPU Utilisation

We also considered the impact of our data hiding method on CPU utilisation. We logged CPU utilization data for the S6 handset in E1 using the Android Device Monitor. We consider encryption and data hiding as two independent processes, since they are separate functions in our source code and can be monitored independently in CPU utilisation analysis.

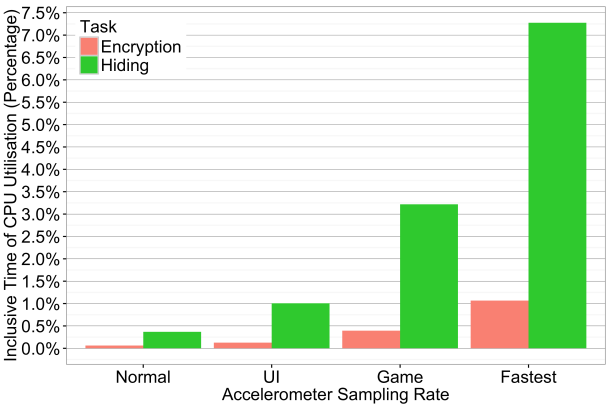


Figure 9. Inclusive time of CPU utilization (%) on S6 handset in E1. Separate utilisation is shown for encryption (red) and hiding (green).

Figure 9 presents the results of encryption vs. hiding at different accelerometer sampling rates. Note that 100% of inclusive CPU time would indicate that the whole period when the data hiding application is running its thread uses a CPU. These results show that our software does not occupy the CPU all the time, meaning that the CPU may set the application thread into the wait state to save energy. We also observe that the CPU was occupied more often with data hiding rather than encryption, even though one call to the data hiding function takes much less time than one call to the encryption function (Figure 6). This disparity is due to the fact that each record of data to be hidden is encrypted once, but requires many calls to the data hiding function, since only 1 or 2 bits can be hidden at a time. For instance, a 32-bit payload is encrypted once but requires 32 calls to the data hiding function when using 2 LSBs (1-bit payload & 1-bit flag).

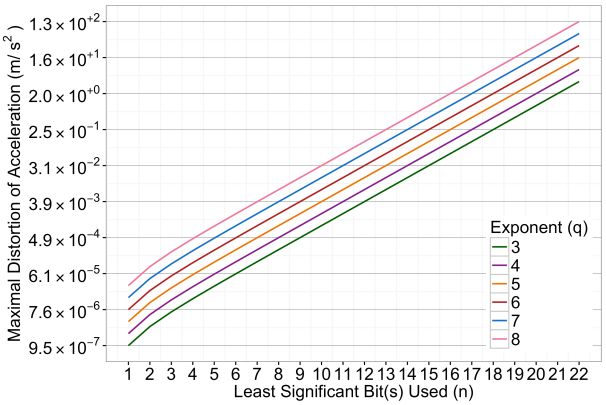


Figure 10. Maximal distortion of acceleration for different LSBs and floating-point exponents. The *y* axis is in base-2 logarithmic scale.

Distortion

According to standards [17], to represent a floating-point number v, the exponent value q in equation (1) must be maximised with the constraint that 2^q is not greater than $|v|$. This means that the amount of error increases as the maximal possible value of $|v|$ is greater. According to the measurement range of common smartphone accelerometer

[25], q is at most 8. Figure 10 depicts the maximal distortion that we theoretically induce for different LSBs and exponents. The number of LSBs (i.e., n) depends on the experiment settings.

We contrast the theoretical prediction with empirical data of the distortion in the carrier signal in E1 and E2 on the S6 handset. The handset was placed on the flat table so that the z axis of accelerometer showed the gravity which was about 10m/s^2, as meaning that a floating point number needs q=3 to represent this value.

In E1 (where 2 LSBs are used) we recorded 2.861×10^{-6}m/s^2 as the maximal absolute value of error in the carrier signal. This result exactly matches our theoretical estimation which is given by equation (2) when n=2 and q=3. Similarly, in E2 (when the number of LSBs was 3), we logged the maximal absolute error 6.676×10^{-6}m/s^2. This also exactly matches our theoretical estimation where equation (2) has n=3 and q=3.

DISCUSSION

Performance

Our results show that smartphone sensor streams can provide sufficiently high capacity for common data hiding scenarios, especially when used with high frequency carrier signals. Depending on the security concerns of smartphone sensing systems, a variety of smartphone data types, such as floating numbers (e.g., magnetometer and GPS), integers (e.g., heart rate) and characters (e.g., human input text and device ID code) can be a suitable payload hosted in the carrier signal.

Indicatively, we measured on the S6 handset a maximum capacity of 26.8B/s with a 1-axis accelerometer carrier signal. Give the expected distortion shown in Figure 10, the capacity for 7 LSBs on a 3-axis carrier signal is 526B/s, with expected distortion between 10^{-5}m/s^2 and 4×10^{-3}m/s^2. To extract this hidden data, a recipient requires knowledge of:

- the data type of the host signal;
- the data type of hidden data;
- the number of LSBs used in the host signal;
- the host signal sorted by timestamp;
- the information for decryption (in the case of AES/GCM, they are the nonce, the authentication tag and the decryption key).

Beyond the confidentiality and integrity provided by AES/GCM encryption, hiding data into another sensor stream obscures the existence of sensitive and private data secret by making it imperceptible. Thus, as Lane et al. [22] have called for, using our approach the type and value of sensitive data streams are not accessible or noticeable to a third party, taking one step closer towards the preservation of privacy. For example, an attacker may find it useful to know that a user is uploading location data, even if they cannot see the actual data. Our method alleviates this concern by obscuring the existence of such sensitive data. In practice, this means that sensitive data is not stored in a separate database field (thus making it perceivable to third parties). In addition, if the payload is an encrypted identity code such as a device ID, it can be used to verify the authenticity of the carrier signal source.

Implementation Issues

Our approach has a manageable computational cost (Figure 9), making it practical for smartphones [18] and allowing power-efficiency OS techniques to reduce its energy footprint, for example setting threads to sleep mode. In addition, the theoretical predictions regarding the distortion caused by our technique (Figure 10) have been empirically confirmed, thus guaranteeing the level of fidelity between the original carrier signal and the signal containing hidden information.

This is important for a range of applications. Certain applications that use accelerometer data require high precision, such as gesture recognition [34], while other applications like scrolling via tilting [5] require crude precision since smartphone accelerometers and gyroscopes produce measurement errors anyway [10]. Our method is flexible enough to account for varying needs regarding the fidelity of processed data, by trading off fidelity and capacity.

Our approach can be adopted by existing sensing systems that already support smartphone sensor data. In particular, we envision that a user with multiple devices (e.g. phone, tablet, smartwatch) would be able to transparently share sensitive between those devices via existing platforms. As long as each device has access to the sensor data, it is possible to extract and decrypt hidden data on the client, without allowing the platform to gain access, or even know that the hidden data exists. This is possible without modifying the platform itself, and not requiring additional "encrypted" fields to be supported.

Medical Sensor Data

Due to its technical characteristics, our proposed data hiding technique can help to address the legislation that many countries have to protect sensitive data, especially medical sensor data [4, 36]. In general, the development of medical information systems has been a challenging and costly affair for many countries [16] due to the complex privacy requirements.

For instance, it is challenging to enable users to retain control of their own data after it has been entered in the system, and giving them access to this data is often a thorny issue [2]. Our method enables users to retain control of their sensitive data even after it has been uploaded on a healthcare information system. For example, during consultation a user could decrypt sensitive information using the secrets stored in their personal device, and show it to the doctor.

Crowdsensing

Additionally, our technique enables the verification of the authenticity or owner of smartphone sensor data. This is particularly relevant to mobile crowdsensing scenarios, either user-driven [6] or agent-driven [24], with diverse application including environmental monitoring and intelligent transportation. In such settings, malicious users or faulty systems can upload tampered or faked data to damage the systems or to defraud benefits if the systems offer rewards for uploading certain data. In this scenario, our technique can offer crucial digital evidence for forensics [28] to ensure the authenticity of smartphone sensor data. For example, this can be achieved by smartphone applications embedding an encrypted unique identification number into every uploaded sensor data stream. When the streams are received, their authenticity can be established by inspecting the identification number. If the received data stream does not contain the ID assigned to a particular client, the systems can consider the data invalid and ignore it.

Along the same lines, initiatives such as Crawdad and Crowdsignals are building up large archives of sensor data. Using our technique, it is possible for users to "physically" embed in this data a unique signature that serves as proof of ownership of the data, and can be used to confirm that no tampering has taken place. The "physical" binding means that even if this sensor data is shared between scientists via email, database services, physical media, and across a variety of file formats, the hidden data persists. This property also ensures that it is "future proof", in the sense that if in the future new ways of sharing data is established, the hidden data will remain available as proof of who owns or generated this sensor stream.

LIMITATIONS AND FUTURE WORK

We have only verified our approach on 6 phones with Android OS 5.0 or above, and we are aware that approximately 65% of Android smartphones run a lower version that 5.0 at time of writing. We expect our method's performance to vary across different handsets, but only in terms of capacity and CPU load. The other features of our method should remain invariant.

Clearly, our method has not been tested on other operating systems, such as Symbian, iOS and Windows, and this would be a crucial next step for our work. A key challenge may be that the implementation of AES/GCM may be unavailable on other handsets, meaning that an ad-hoc algorithm may be needed. Although developers can employ other encryption algorithms, this may downgrade the performance and security level. Another technical issue is that the computational efficiency in other handsets environments can be significantly lower than Android 5.0, meaning that they cannot use high-frequency sensor data as the carrier signal.

In addition, we have not tested our method with a broad range of external sensors or devices (such as smartwatches). Platforms with higher constraints (such as smartwatches) may find it challenging to attain high capacity data hiding.

During the selection of carrier signals, objectively quantifying the sensitivity of each sensor can provide greater robustness to the data hiding mechanism. This requires a substantial body of future work. The positions of sensors (i.e., their sensitivity) in Figure 1 may depend on various factors, such as social context, network environments and capabilities of attackers.

Our future work will also include a mechanism to balance the tradeoff between capacity and fidelity in carrier signals. A large number of LSBs leads to high capacity and low fidelity in the carrier signals. Therefore, this mechanism should adaptively identify a suitable upper bound of LSBs in different types of data hiding scenarios.

CONCLUSION

We propose a data hiding method to embed sensitive information into smartphone sensor data streams. Our method combines encryption with data hiding, and can be adopted by smartphone sensing systems to secure sensitive data or to prove the authenticity of data. Due to the imperceptibility of data hiding techniques, an unauthenticated party does not notice the type and value of hidden sensitive data stream by interception, thus alleviating some of the privacy problems of smartphone sensing systems mentioned in literature [22].

We evaluated with a variety of handsets, data types, and settings. Our experimental results show that it is feasible to encrypt and embed common types of smartphone data (e.g., magnetometer readings, heart rate, GPS location, human input text and device identification code) into high-frequency sensor streams, such as accelerometer, in real time. Moreover, we show that AES/GCM encryption and data hiding operations have manageable impact on the CPU utilisation of the sensing application thread, meaning that our approach will not be bottlenecked by resource-constrained environments of smartphones. We demonstrate that our approach is able to maintain high fidelity after data hiding, and can provide strong guarantees regarding fidelity by adjusting the number of LSBs used for hiding. Our findings make this data hiding method attractive for smartphones sensing systems that collect sensitive data or require high data authenticity, such as medical systems and digital forensics applications.

ACKNOWLEDGMENTS

This work is partially funded by Infotech Oulu, TEKES-funded VitalSens project, the Academy of Finland (Grants 276786-AWARE, 285062-iCYCLE, 286386-CPDSS, 285459-iSCIENCE), and the European Commission (Grants PCIG11-GA-2012-322138, 645706-GRAGE, and 6AIKA-A71143-AKAI).

REFERENCES

1. Gregory D. Abowd. 2012. What next, ubicomp? Celebrating an intellectual disappearing act. In

Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12, 31-40. http://dx.doi.org/10.1145/2370216.2370222.

2. Corey M. Angst and Ritu Agarwal. 2009. Adoption of Electronic Health Records in the Presence of Privacy Concerns: The Elaboration Likelihood Model and Individual Persuasion. *MIS Q.* 33, 2: 339-370.
3. Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge J. L. Reyes-Ortiz. 2013. A public domain dataset for human activity recognition using smartphones. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN*.
4. David Blumenthal and Marilyn Tavenner. 2010. The meaningful use regulation for electronic health records. *New England Journal of Medicine* 363, 6: 501-504.
5. Sebastian Boring, Marko Jurmu and Andreas Butz. 2009. Scroll, Tilt or Move It: Using Mobile Phones to Continuously Control Pointers on Large Public Displays. In *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7*, ACM, 161-168. http://dx.doi.org/10.1145/1738826.1738853.
6. Yohan Chon, Nicholas D. Lane, Yunjong Kim, Feng Zhao and Hojung Cha. 2013. Understanding the Coverage and Scalability of Place-centric Crowdsensing. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ACM, 3-12. http://dx.doi.org/10.1145/2493432.2493498.
7. G Coatrieux, L Lecornu, B Sankur and Ch Roux. 2006. A Review of Image Watermarking Applications in Healthcare. In *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, IEEE, 4691-4694. http://dx.doi.org/10.1109/IEMBS.2006.259305.
8. Christian Collberg and Jasvir Nagra. 2009. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection.* Addison-Wesley Professional.
9. John P. Collomosse and Tim Kindberg. 2008. Screen Codes: Visual Hyperlinks for Displays. In *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications*, ACM, 86-90. http://dx.doi.org/10.1145/1411759.1411782.
10. Ionut Constandache, Xuan Bao, Martin Azizyan and Romit R. R. Choudhury. 2010. Did You See Bob?: Human Localization Using Mobile Phones. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, ACM, 149-160. http://dx.doi.org/10.1145/1859995.1860013.
11. Ingemar J. Cox, Joe Kilian, Tom Leighton and Talal Shamoon. 1996. A secure, robust watermark for multimedia. In *Information Hiding* (eds.). Springer Berlin Heidelberg, 185-206.
12. Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich and Ton Kalker. 2008. *Digital Watermarking and Steganography.* Morgan Kaufmann Publishers Inc..
13. Stefan Dernbach, Barnan Das, Narayanan C. Krishnan, Brian L. Thomas and Diane J. Cook. 2012. Simple and Complex Activity Recognition through Smart Phones. In *International Conference on Intelligent Environments*, IEEE, 214-221. http://dx.doi.org/10.1109/IE.2012.39.
14. Morris J. Dworkin. 2007. *SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.*
15. Frank Hartung and Martin Kutter. 1999. Multimedia watermarking techniques. *Proceedings of the IEEE* 87, 7: 1079-1107. http://dx.doi.org/10.1109/5.771066.
16. Richard Heeks. 2006. Health information systems: Failure, success and improvisation. *International Journal of Medical Informatics* 75, 2: 125-137. http://dx.doi.org/10.1016/j.ijmedinf.2005.07.024.
17. 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*: 1-70. http://dx.doi.org/10.1109/IEEESTD.2008.4610935.
18. Hussam Juma, Ibrahim Kamel and Lami Kaya. 2008. Watermarking sensor data for protecting the integrity. In *International Conference on Innovations in Information Technology*, IEEE, 598-602. http://dx.doi.org/10.1109/INNOVATIONS.2008.4781662.
19. Apu Kapadia, David Kotz and Nikos Triandopoulos. 2009. Opportunistic sensing: Security challenges for the new paradigm. In *Communication Systems and Networks and Workshops*, IEEE, 1-10. http://dx.doi.org/10.1109/COMSNETS.2009.4808850.
20. Predrag Klasnja, Sunny Consolvo, Tanzeem Choudhury, Richard Beckwith and Jeffrey Hightower. 2009. Exploring Privacy Concerns About Personal Sensing. In *Proceedings of the 7th International Conference on Pervasive Computing*, Springer-Verlag, 176-183. http://dx.doi.org/10.1007/978-3-642-01516-8_13.
21. Vassilis Kostakos and Denzil Ferreira. 2015. The Rise Of Ubiquitous Instrumentation. *Frontiers in ICT* 2, 3: 1-2. http://dx.doi.org/10.3389/fict.2015.00003.
22. Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury and Andrew T. Campbell. 2010. A Survey of Mobile Phone Sensing. *Communications Magazine, IEEE* 48, 9: 140-150. http://dx.doi.org/10.1109/MCOM.2010.5560598.

23. Reed Larson and Mihaly Csikszentmihalyi. 2014. *The Experience Sampling Method.* Springer Netherlands.
24. Teemu Leppänen, José A. Lacasia, Yoshito Tobe, Kaoru Sezaki and Jukka Riekki. 2015. Mobile crowdsensing with mobile agents. *Autonomous Agents and Multi-Agent Systems*: 1-35. http://dx.doi.org/10.1007/s10458-015-9311-7.
25. LIS3DH MEMS digital output motion sensor ultra low-power high performance 3-axes "nano" accelerometer. http://www.st.com/web/en/resource/technical/document/datasheet/CD00274221.pdf.
26. Cristina V. Lopes and Pedro M. Q. Aguiar. 2003. Acoustic Modems for Ubiquitous Computing. *IEEE Pervasive Computing* 2, 3: 62-71. http://dx.doi.org/10.1109/MPRV.2003.1228528.
27. Nai Miao, Yutao He and Jane Dong. 2012. hymnMark: Towards Efficient Digital Watermarking on Android Smartphones. In *Proceedings of the International Conference on Wireless Networks (ICWN)*, 1-8.
28. Alexios Mylonas, Vasilis Meletiadis, Lilian Mitrou and Dimitris Gritzalis. 2013. Smartphone sensor data as digital evidence. *Computers & Security* 38: 51-75. http://dx.doi.org/10.1016/j.cose.2013.03.007.
29. W. Pan, G. Coatrieux, J. Montagner, N. Cuppens, F. Cuppens and C. Roux. 2009. Comparison of some reversible watermarking methods in application to medical images. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, IEEE, 2172-2175. http://dx.doi.org/10.1109/IEMBS.2009.5332425.
30. Jennifer Pearson, Simon Robinson, Matt Jones, Amit Nanavati and Nitendra Rajput. 2013. ACQR: Acoustic Quick Response Codes for Content Sharing on Low End Phones with No Internet Connectivity. In *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services*, ACM, 308-317. http://dx.doi.org/10.1145/2493190.2493195.
31. Yanzhi Ren, Yingying Chen, Mooi C. Chuah and Jie Yang. 2015. User Verification Leveraging Gait Recognition for Smartphone Enabled Mobile Healthcare Systems. *Mobile Computing, IEEE Transactions on* 14, 9: 1961-1974. http://dx.doi.org/10.1109/TMC.2014.2365185.
32. Amit Sahai. 2004. Secure Protocols for Complex Tasks in Complex Environments. In *Progress in Cryptology - INDOCRYPT 2004*, Springer Berlin Heidelberg, 14-16. http://dx.doi.org/10.1007/978-3-540-30556-9_2.
33. Frank Y. Shih. 2007. *Digital Watermarking and Steganography Fundamentals and Techniques*. CRC Press.
34. Boris Smus and Vassilis Kostakos. 2010. Running gestures: hands-free interaction during physical activity. In *International Conference on Ubiquitous Computing Adjunct*, ACM, 433-434. http://dx.doi.org/10.1145/1864431.1864473.
35. Ryohei Suzuki, Daisuke Sakamoto and Takeo Igarashi. 2015. AnnoTone: Record-time Audio Watermarking for Context-aware Video Editing. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ACM, 57-66. http://dx.doi.org/10.1145/2702123.2702358.
36. Astrid M. van Ginneken. 2002. The computerized patient record: balancing effort and benefit. *International Journal of Medical Informatics* 65, 2: 97-119. http://dx.doi.org/10.1016/s1386-5056(02)00007-2.
37. R. Velumani and V. Seenivasagam. 2010. A reversible blind medical image watermarking scheme for patient identification, improved telediagnosis and tamper detection with a facial image watermark. In *IEEE International Conference on Computational Intelligence and Computing Research*, IEEE, 1-8. http://dx.doi.org/10.1109/ICCIC.2010.5705832.
38. Honggang Wang, Dongming Peng, Wei Wang, Hamid Sharif and Hsiao-Hwa Chen. 2008. Energy-Aware Adaptive Watermarking for Real-Time Image Delivery in Wireless Sensor Networks. In *International Conference on Communications*, IEEE, 1479-1483. http://dx.doi.org/10.1109/ICC.2008.286.
39. Wei Zhang, Yonghe Liu, Sajal K. Das and Pradip De. 2008. Secure data aggregation in wireless sensor networks: A watermark based authentication supportive approach. *Pervasive and Mobile Computing* 4, 5: 658-680. http://dx.doi.org/10.1016/j.pmcj.2008.05.005.
40. Wu Zhou, Xinwen Zhang and Xuxian Jiang. 2013. AppInk: Watermarking Android Apps for Repackaging Deterrence. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ACM, 1-12. http://dx.doi.org/10.1145/2484313.2484315.

Chapter 5

Validating machine learning design

5.1 Aims and hypotheses

This chapter investigates how the machine learning design of context-driven mobile applications can be efficiently and systematically validated in the laboratory. We proposed a validation approach that aims at the following aspects:

1. **Classification or regression.** Our approach focuses on two machine learning tasks: classification and regression. They are the most common tasks for the development of mobile applications [82,92]. We show that either of the two can be useful in some scenarios. For example, to predict when a mobile application will be used by the user, a regression algorithm can be trained to output the numerical result (e.g., 5 minutes) using contextual data. Also, a classifier can be trained to give a positive or negative answer by considering whether the user will use the mobile application in next 5 minutes. Although slightly different in the considered questions, classification and regression are both useful in this case.

2. **Performance of different algorithms.** For both classification and regression, our approach compares the performance of different algorithms. Among all classification algorithms, our approach selects Naive Bayes, J48 Decision Tree, Random Forests [24] and Support Vector Machines (SVM) [31], since they are widely used in previous research of mobile applications (e.g., [121–123]). Also, they can easily be evaluated or implemented using the machine learning analysis tool Weka [66] which has a cross-platform Java library for application development. Likewise, for regression, since the target function in prediction can be either linear or nonlinear, our approach selects linear

regression and Random Forests. For both classification and regression, our approach selects Random Forests since it provides feature importance measures to evaluate how helpful each feature is in prediction.

3. **Feature importance.** To quantify the importance of features in classification, our approach relies on two kinds of measures: the "select attributes" function of Weka, and the feature importance of Random Forests. The "select attributes" function computes the mean ranking of each feature by measuring the accuracy decrease caused by removing a subset of features in classification. The search method of feature subsets is Greedy Stepwise which searches forward or backward in a greedy manner within all possible feature subsets. Random Forests have two feature importance measures that are generated in the training process: Mean Decrease Accuracy (MDA) and Mean Decrease Impurity (MDI). MDA is calculated by randomly permuting features with out-of-bag samples. MDI is the mean weighted impurity decrease for all nodes containing a feature [24]. For regression, our approach adopts two measures of Random Forests: increased mean squared error (IncMSE) and increased node purity (IncNodePurity). IncMSE measures the increase of mean squared error by permuting features with out-of-bag samples. IncNodePurity computes the mean increase in purity of all nodes containing a feature within the trees [55].

4. **Software-generated features.** As mobile devices have limited battery capacity, our approach evaluates the prediction performance of models without features generated by hardware sensors which are often power-hungry.

5. **Personalised models or cold start.** Our approach compares personalised models and cold start models [136]. Personalised models are trained using data collected on the mobile device of each individual user. The user of a personalised model is supposed to be the one who generates the training data for this model. In contrast, cold start models are trained using data collected on the mobile devices of all other users (in practice, cold start models learn from data collected from all previous users). The user of a cold start model is supposed to be a new user who has never generated any training data for this model.

6. **Energy consumption.** Our approach quantifies energy consumption of continuous sensing and prediction on a physical device.

7. **Design phase testing in the laboratory.** Our approach aims to validate the design choices of machine learning applications at the design phase, without implemented software or with the minimal efforts of implementing a software prototype. For some daily-life scenarios such as long-term phone usage, ground-truth labels with context data can only be obtained in the real world, rather than in the laboratory. Our approach first conducts data collection with real users in a in-the-wild study. Then, based on the collected data, our approach intends to perform all other tests in the laboratory.

We hypothesised that our approach can achieve efficient and systematic laboratory-based validation for the machine learning design of context-driven mobile applications. With an exemplar mobile application that uses machine learning and contextual data to predict when a user will unlock its smartphone, our approach is detailed in the attached publication in Section 5.2.

5.2 Publication

Chu Luo, Aku Visuri, Simon Klakegg, Niels van Berkel, Zhanna Sarsenbayeva, Antti Möttönen, Jorge Goncalves, Theodoros Anagnostopoulos, Denzil Ferreira, Huber Flores, Eduardo Velloso, and Vassilis Kostakos. Energy-efficient prediction of smartphone unlocking. *Personal and Ubiquitous Computing*, 23(1), pages 159-177, 2019. `https://doi.org/10.1007/s00779-018-01190-0`.

Noname manuscript No.
(will be inserted by the editor)

Energy-Efficient Prediction of Smartphone Unlocking

Chu Luo · Aku Visuri · Simon Klakegg · Niels van Berkel · Zhanna Sarsenbayeva · Antti Möttönen · Jorge Goncalves · Theodoros Anagnostopoulos · Denzil Ferreira · Huber Flores · Eduardo Velloso · Vassilis Kostakos ·

Received: date / Accepted: date

Abstract We investigate the predictability of the next unlock event on smartphones, using machine learning and smartphone contextual data. In a two-week field study with 27 participants, we demonstrate that it is possible to predict when the next unlock event will occur. Additionally, we show how our approach can improve accuracy and energy efficiency by solely relying on software-related contextual data. Based on our findings, smartphone applications and operating systems can improve their energy efficiency by utilising short-term predictions to minimise unnecessary executions, or launch computation-intensive tasks, such as OS updates, in the locked state. For instance, by inferring the next unlock event, smartphones can pre-emptively collect sensor data or prepare timely content to improve the user experience during the subsequent phone usage session.

Keywords Smartphones · Machine learning · Sensors · Context-awareness

1 Introduction

Predicting when a phone will be unlocked in an accurate and energy-efficient manner is crucial to both resource management and user experience. Literature makes a distinction between glancing at the phone's lock screen [3] versus actually unlocking the device [5]. We are interested in predicting the latter, *i.e.* when users actually choose to unlock their device for further use. Although a plethora of work contributes to understanding phone usage patterns, little work has considered techniques to predict smartphone unlock

C. Luo, N. van Berkel, Z. Sarsenbayeva, J.Goncalves, E. Velloso, V. Kostakos
The University of Melbourne, Australia
C. Luo E-mail: CHUL3@student.unimelb.edu.au
V. Kostakos E-mail: vassilis.kostakos@unimelb.edu.au
A. Visuri, S. Klakegg, A. Möttönen, D. Ferreira
University of Oulu, Finland
T. Anagnostopoulos
University of West Attica, Greece
H. Flores
University of Helsinki, Finland

events. Such predictions can benefit smartphones in a number of ways, such as:

1. *trigger timely data acquisition*: prediction of upcoming unlocking events allows mobile applications, especially lockscreen applications, to prepare timely content which users can see right after unlocking. This can directly improve the user experience on the device. A typical use case is lockscreen-based crowdsourcing, which presents questions with pictures or icons for users to answer via various unlock gestures. Given a prediction of the next upcoming unlocking event, smartphones can, in advance, connect to the Internet to download timely questions and their media files. Without this prediction, the user experience may suffer when users see obsolete questions or have to wait for the newest content from the ad-hoc downloading.
2. *disable unnecessary executions*: given a prediction that the phone is likely to remain locked for a certain time, the phone can suspend the execution of applications that continuously prepare content. For example, mobile crowdsourcing applications, such as Truong et al.'s, collect sensor data and download online content every few minutes[43]. The continuous execution of such applications consumes a significant amount of power, which can prove detrimental to the user experience [33] .
3. *schedule computation-intensive tasks*: when predicting that the phone is likely to remain locked, the operating system can launch computation-intensive background tasks, such as data synchronisation or updates, which would have a negative impact on the user experience during phone use. A typical example is the OS update which often involves hundreds of megabytes of data to download, and a long reboot. During an OS update, smartphone users must wait for the completion without any possibility for interaction. Since untimely schedules of such computation-intensive tasks greatly degrade user experience, Apple's iOS 10 engineers have publicly highlighted that scheduling OS updates is a challenging problem [29].

Crucially, the energy cost to achieve such predictions must be sufficiently low, otherwise making these predictions will defeat their very purpose. Therefore energy efficiency is an essential requirement in the selection of smartphone context features and the prediction process.

We hypothesise that by analysing smartphone context we can develop a learning model to predict the next unlock event. We intend to make this prediction regardless of whether the user actively unlocks their phone, or the user is notified to unlock it by a notification. In a 2-week field study, we used the AWARE framework [17] to collect phone usage and context from 27 users during their daily activities. Our analysis is multi-faceted, and a key tradeoff that we investigate is between prediction accuracy and time windows. Overall, our work makes these contributions:

1. We present an energy-efficient approach to predict the next unlocking event on a smartphone, by considering smartphone unlock time prediction as either a regression problem or a binary classification.
2. We identify which features are the strongest indicators of upcoming unlock events, and show that software-based context data be more useful predictors than hardware sensors.
3. We describe and discuss the trade-off between the accuracy of positive and negative predictions in the selection of time windows for this classification.

4. We evaluate two real-world deployment scenarios: using personalised models and cold start.

2 Related Work

Predicting when applications will be launched helps to reduce launch time and improves efficiency, particularly for frequent users [45,39]. Shin et al. dynamically presented application icons on the launch screen by predicting application usage through contextual data with accurate predictions in nearly 90% of cases [39]. Lee et al. reduced the size of contextual data needed for such predictions by integrating a feature selection algorithm [31].

These techniques focus on either expanding the functionality of the lock screen or predicting interactions following the unlock event. No study has investigated the accuracy of predicting when the smartphone will be unlocked.Predicting the time when users will unlock their phones is beneficial for applications related to phone unlocking, such as smart notifications [38], music recommendation, and auto-execution services [31].

2.1 Smartphone Usage Patterns

Several factors impact phone usage patterns, including time, location, and mental state. Time is an important measure in previous studies of smartphone usage patterns. In a longitudinal study, Yan et al. showed that most phone interactions are very short, 50% of interactions last less than 30 seconds, and 90% last less than 4 minutes [46]. Approximately 40% of all application uses consist of short bursts of up to 15 seconds, particularly when the user is at home and alone—what Ferreira et al. call *micro-usage* [16]. Falaki et al. found that phone interactions happen very frequently (10-200 times per day), are short (10-250 seconds), and involve multiple applications (10-90) [13]. McGregor et al. identified three usage patterns in a study with 15 iPhone users: micro-breaks, digital knitting, and mobile reading [8]. Van Berkel et al. further analysed smartphone usage sessions and recognised that multiple micro-breaks can combine into longer combined sessions [5].

In terms of correlation between phone usage and user location, Verkasalo [44] highlights that more than half of the time spent on smartphones is at home; over 25% on the move; and the remainder at work. In a case study of a Chinese city, Yuan et al. found that mobile phone usage correlates with human travel behaviour [47]. One important finding is that when users make more phone calls, their average movement radius increases. Furthermore, based on a smartphone usage dataset from 77 participants over a 9-month period, Do et al. report that application usage is strongly correlated with locations [12]. For example, voice calls are frequently made at work. In contrast, clock application use occurs very seldom at work.

In addition, researchers found that phone usage is impacted by mental states. For example, several studies state that when people feel bored, mobile phones commonly serve as means to kill time [8,34,36]. Other studies found links between phone usage and compulsive anxiety [27], depressive symptons, higher interpersonal anxiety, and lower self-esteem [22].

Although these previous studies illuminate various aspects of smartphone usage patterns, there is little insight into what contextual factors lead to the user actually unlocking their phone.

2.2 Context-Aware Services on Smartphones

With embedded sensors and increasingly capable processors, modern smartphones act as more than a communication tool. New applications for e-health, environmental monitoring and transportation benefit from the collection of big sensor data on smartphones [30]. Particularly, providing context-aware services to smartphone users is of increasing interest to the HCI and UbiComp community. Studies have repeatedly confirmed the feasibility of inferring user intent using contextual data on smartphones. For example, predicting whether users are available to answer an incoming call[35], how responsive they are to instant messaging [2,37] and when to best interrupt users with notifications [18,38].

However, most of these context-aware services are battery-intensive as they rely on continuous sensing [33,32], leading to the need for daily charging [14,15]. The literature points to alternative ways of using sensor data, such as relying on low-energy sensors wherever possible [4], and avoiding energy-intensive sensors like GPS [48]. To reduce the power cost from context-aware services, Lu et al. proposed the Jigsaw continuous sensing engine [33]. Jigsaw adaptively controls accelerometer, microphone and GPS sensors according to phone sensing environments. Similarly, Chon et al. presented a smartphone sensing manager SmartDC that monitors, learns, and predicts user mobility and location to conduct adaptive energy-efficient duty cycling [11].

In spite of these efforts, prior work has not attempted to achieve energy efficiency by minimising irrelevant executions while the user keeps the phone in the locked state for a long time. If a smartphone application can predict when it will be unlocked next, it is easier for other applications to prepare relevant content and schedule tasks for the next session, such as delaying or dismissing context-aware notifications, downloading timely content, or changing phone settings (e.g., slowing down data synchronisation and enabling power saving mode).

3 Methodology

We conducted an in-the-wild study with 27 Android smartphone users (16M/11F) recruited through mailing lists of at our University. The data collection lasted for two weeks. Most participants were students or had an academic background. Participants were young (M = 24.7, SD = 4.32), and from various academic backgrounds with 14 participants from engineering, 6 from humanities, 2 from business, 1 from natural sciences, and three preferred to not say. Most participants were local (20) to our city, while seven were foreign.

After we explained the data collection mechanism and study process, we installed the software on each participant's phone. We then instructed participants to use their smartphones as they normally would in their everyday life. Our software did not require any active input from participants, nor did it notify them in any way. Our software logged smartphone usage events, including unlocking events and contextual data from sensors. The application

Table 1: Features and data types extracted from the smartphone sensors and other data sources used in data collection.

Data Source	Description	Feature and Data Type
Activity	Physical activity	e.g. walking, running and being in vehicle (value within a categorical set)
Application	Foreground application	The number of foreground applications used in last session (integer)
Data Traffic	Data upload and download transmission	Since last session, WiFi data upload (integer); WiFi data download (integer); Cellular data upload (integer); Cellular data download (integer);
Demographics	Gender	Gender (value within a categorical set)
Light	Ambient light (lux)	Ambient light lux (integer)
Location	Location on, off and mode (GPS, network)	GPS availability (Boolean); Network location availability (Boolean);
Pedometer	Steps travelled	Step count since last session (integer)
Proximity	Coverage on phone	Proximity (Boolean)
Screen Events	Screen on, off and unlocking	Duration of last session (integer); time since last session (integer)
Time	Timestamp of phone clock	Minute of hour (integer); hour of day (integer); day of week (integer)
WiFi	WiFi on, off and the number of nearby spots	WiFi availability (Boolean); WiFi spots nearby (integer);

worked as a plugin of the AWARE framework, an open-source middleware to capture contextual data on mobile devices [17]

3.1 Data Collection and Features

Our plugin constantly collected contextual data in the background. Table 1 describes the features extracted from the sensors and other data sources. We consider a *session* of phone usage as a continuous interaction period from the moment of unlocking the smartphone to the moment of the subsequent locked/power off state. To minimise power consumption, only low-frequency proximity (200ms/sample), light (5s/sample), location (180s/GPS sample, or 300s per network sample when GPS unavailable) and WiFi (60s/scan) sensors are logged. We also used Google's Activity Recognition API (6s/sample) [20] and Android's Pedometer API [21] to obtain users' activities and steps, computed from accelerometer data. Based on such contextual data, we extracted 18 features. In addition, we also considered gender as a feature, as reported by participants.

Because the characteristics of a previous usage session might be informative as to the timing of the next unlocking event, we collect application usage as provided by the operating system. Finally, data traffic reveals the intensity of networked activities, as measured by two hardware modules (WiFi and Cellular antenna).

3.2 Analysis

Our analysis first considers unlock time prediction as a regression problem: given the user context, the algorithm should predict the timing of when the user will unlock the phone. Without knowing the linearity of the relation, we compare the performance of two regression algorithms: multiple linear regression as a linear approach (it serves as a baseline of regression), and Random Forests as a nonlinear approach (using Weka's default parameters, i.e., Random Forests with 100 trees and $\lfloor \log_2 m \rfloor + 1$ features, where m is the number of initial features). For Random Forests, using $\lfloor \log_2 m \rfloor + 1$ features avoids high complexity of the model and reduce overfitting. Considering that there are only 19 features in our dataset ($\lfloor \log_2 19 \rfloor + 1 = 5$ features per tree), 100 trees in Random Forests are sufficient. We also evaluate the performance loss incurred in using solely low-power software-related features as compared to the full feature set. Finally, we compare the performance of personalised models to cold start models (i.e. models trained on other users' data) [6].

We also attempt to use a classifier approach: given the user context, the classifier should predict whether the user will unlock the phone within the next x minutes. Similar to previous work employing different machine learning algorithms on smartphones [35,38], we compared four widely used classifiers to run 10-fold cross-validation tests: Naive Bayes, J48 Decision Tree, Random Forests and Support Vector Machine (SVM) with the Radial Basis Function (RBF) kernel (with normalised samples). The parameters were left to their default values in Weka for simplicity, i.e., Random Forests with 100 trees and $\lfloor \log_2 m \rfloor + 1$ features (m is the number of initial features); SVM with the RBF kernel ($\gamma = \frac{1}{m+1}, C = 1$).

In summary, our analysis empirically investigates the following questions:

1. How well do machine learning algorithms perform?
2. What are the most useful features in the prediction?
3. How does performance change using low-power software-related data sources?
4. How does a personalised model perform on each user who has provided training data?
5. How does a general model trained by previous users' data perform on a new user who has not provided training data (the new user cold start problem [6])?
6. How much energy does the prediction consume on a real device?

3.3 Measures

For regression, we used three commonly used measures: coefficient of determination (R^2), mean absolute error (MAE) and square root of mean squared error (RMSE). For the importance of each feature in the regression model, we used two feature importance measures provided by Random Forests: increased mean squared error (IncMSE) and increased node purity (IncNodePurity). For each feature, IncMSE is computed by comparing the difference of mean squared error in predictions after permuting this feature. IncNodePurity corresponds to the increase in node purity of having each feature within the trees related to the mean squared error. Hence, a high IncNodePurity of a feature means that this feature significantly helps the

Table 2: Confusion matrix for classification analysis of the two classes.

	Actual Unlock (Class "Yes")	Actual Lock ("No")
Predicted Unlock (Class "Yes")	TP	FP
Predicted Lock (Class "No")	FN	TN

Table 3: Recall and Precision definitions of the two classes.

	Recall	Precision
Unlock (Class "Yes")	$\frac{TP}{TP+FN}$	$\frac{TP}{TP+FP}$
Lock (Class "No")	$\frac{TN}{TN+FP}$	$\frac{TN}{TN+FN}$

algorithm to reduce mean squared error. The details of these measures are illustrated by Genuer et al. [19].

In the classification models, because of class imbalance, we report ROC area, precision, and recall, instead of accuracy, as recommended by Kostakos et al. [28]. All results are reported using a 10-fold cross-validation on the entire dataset. In our data we separately consider the two classes: "no" (the cases where the user did not unlock the phone within x minutes), and "yes" (where the user did unlock the phone within x minutes). Table 2 shows the confusion matrix for classification analysis on the two classes. Based on the confusion matrix, Table 3 shows recall and precision definitions of the two classes.

For the feature importance of classification, we used Mean Decrease Accuracy (MDA) and Mean Decrease Impurity (MDI). MDA is measured by randomly permuting features in the out-of-bag samples, while MDI (also known as Mean Decrease Gini if the Gini index is used as in our analysis) is the average weighted impurity decreases for all nodes containing a specific feature across all trees [7]. Specifically, we used 3 types of MDA to investigate the influence of removing each feature on both classes, as well as the overall prediction. The higher the MDA value is, the more importance a feature has. In practice, the scenario may require high accuracy for only one class.

In contrast, MDI does not measure feature importance for each class. MDI provides an overall assessment of each feature's importance based on the assumption that if a feature has high usefulness, it tends to divide nodes with multiple classes into nodes with single classes. However, this assumption may result in bias. Usually, feature importance analysis should use a combination of different measures to avoid the bias from a single measure [41].

4 Results and Optimisation

We collected 7,472,609 entries of raw user data, from which we sampled 175,684 segments of phone usage. We used Weka [23] to analyse these samples in both the regression and classification problems.

4.1 Regression Approaches

Table 4 shows the regression results using 3 commonly used measures: coefficient of determination (R^2), mean absolute error (MAE) and square root of

Table 4: Regression results of the entire dataset in coefficient of determination (R^2), mean absolute error (MAE) (in minutes) and square root of mean squared error (RMSE) (in minutes).

Dataset	Regression type	R^2	MAE	RMSE
All sensors	Linear Regression	0.06	122.59	263.96
All sensors	Random Forests	0.98	14.24	48.09
Software sensors only	Linear Regression	0.03	124.87	268.11
Software senors only	Random Forests	0.98	12.50	45.85

Table 5: Regression results of personalised models and the model of cold start in coefficient of determination (R^2), mean absolute error (MAE) (in minutes) and square root of mean squared error (RMSE) (in minutes).

Bootstrap approach	Regression type	R^2	MAE	RMSE
Personalised	Linear Regression	0.08	117.16	181.51
Personalised	Random Forests	0.94	8.89	24.44
Cold Start	Linear Regression	0.04	168.80	251.87
Cold Start	Random Forests	0.96	150.05	258.40

mean squared error (RMSE). Times are displayed in minutes. Results show that for both algorithms, removing hardware features does not lead to a significant performance loss. Also, the Random Forests algorithm outperforms linear regression in both accuracy and dispersion.

Table 5 shows the regression results for the personalised and cold start models. Although Random Forests significantly outperformed linear regression in personalised models, the accuracy and dispersion are similarly poor for the cold start model.

To quantify the relative importance of each feature when using regression, we used two feature importance measures provided by Random Forests: IncMSE and IncNodePurity. The unit of IncMSE and IncNodePurity is the same as the regression process: min^2. Table 6 shows the importance of all the 19 features using these two measures. The 5 features with the highest IncMSE are: WiFi data download, duration of last session, time since last session, WiFi data upload, number of foreground apps used in last session. Three of these features come from software sensors. The 5 features with the highest IncNodePurity are: duration of last session, time since last session, WiFi data upload, WiFi data download, hour of day. Again, 3 of these features come from software sensors. The difference of top features across the two measures is little, meaning that the results have high robustness.

Table 7 shows the correlation (Spearman's rank correlation coefficients) between each feature and the time before next unlocking event. We only considered numerical features in this analysis (10 out of 19 features). We found that all 10 showed a significant ($\alpha = 0.05$) correlation with the time before the next unlocking event, although relations were mostly weak. The time since last session, a software-related feature, had the strongest relation ($r_s = 0.411$). From the aspect of correlation, the results reflected the different importances of each factor. It is worth noting that features with weak relations may also be useful if the classification or regression method can learn from nonlinear data.

Table 6: When using Random Forests regression, the importance (in min^2) of all the features is estimated by two measures (IncMSE and IncNodePurity)

Feature	IncMSE	IncNodePurity
Time since last session	65104.8	13.51 $\times 10^8$
Duration of last session	68947.3	23.67 $\times 10^8$
Hour of day	51155.7	10.17 $\times 10^8$
Activity	7382.5	1.12 $\times 10^8$
Minute of hour	1970.9	1.28 $\times 10^8$
Day of week	32953.2	8.31 $\times 10^8$
Proximity	17930.7	2.75 $\times 10^8$
Ambient light	34608.7	3.73 $\times 10^8$
Cellular data download	24750.8	4.99 $\times 10^8$
Cellular data upload	32317.6	7.28 $\times 10^8$
WiFi data download	78213.3	11.04 $\times 10^8$
WiFi data upload	64162.6	11.09 $\times 10^8$
Step count since last session	19901.4	2.13 $\times 10^8$
Number of foreground apps used in last session	55934.8	5.51 $\times 10^8$
GPS availability	13162.1	1.22 $\times 10^8$
WiFi spots nearby	13057.4	2.67 $\times 10^8$
Network location availability	10391.2	2.29 $\times 10^8$
WiFi availability	7995.7	1.23 $\times 10^8$
Gender	18829.9	2.31 $\times 10^8$

Table 7: Correlation between each feature and the time before next unlocking event, measured by Spearman's rank correlation coefficients.

Feature	p-value	r_s
Time since last session	$< 2.220 \times 10^{-16}$	0.411
Duration of last session	$< 2.220 \times 10^{-16}$	0.024
Hour of day	N/A	N/A
Activity	N/A	N/A
Minute of hour	N/A	N/A
Day of week	N/A	N/A
Proximity	N/A	N/A
Ambient light	$< 2.220 \times 10^{-16}$	-0.182
Cellular data download	1.548×10^{-10}	-0.015
Cellular data upload	0.042	0.005
WiFi data download	$< 2.220 \times 10^{-16}$	0.136
WiFi data upload	$< 2.220 \times 10^{-16}$	0.153
Step count since last session	$< 2.220 \times 10^{-16}$	-0.214
Number of foreground apps used in last session	$< 2.220 \times 10^{-16}$	-0.098
GPS availability	N/A	N/A
WiFi spots nearby	$< 2.220 \times 10^{-16}$	-0.072
Network location availability	N/A	N/A
WiFi availability	N/A	N/A
Gender	N/A	N/A

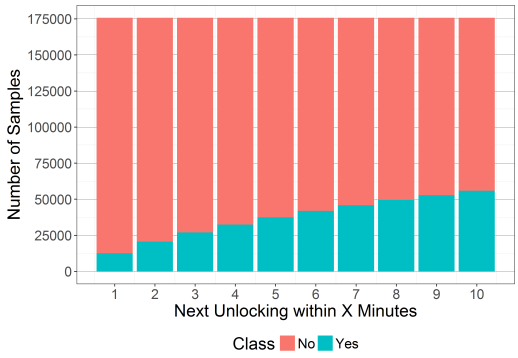


Fig. 1: 175,684 samples of the absence and presence of unlocking events measured by different time windows at different moments during our study.

4.2 Classification Approaches

When considering the average number of unlocking events our users exhibited per day, the maximum was 163, which is in line with previous studies [13]. Given that phone unlocking events can be rather frequent during the day, and that mobile applications can download sufficiently large files in several minutes via either WiFi or 4G, we decided to also model our problem as a classification rather than regression.

Doing so means that we do not try to answer the question "When will the next unlock happen?", but rather we ask the question "Will the next unlock happen in the next x minutes?" In essence, this is an easier question to answer, but it stil provides utility given that many smartphone operations can complete within a few minutes.

In our analysis, we consider a time period of up to 10 minutes, in 1-minute segments. Therefore, our classifier tries to answer the question: "Will the next unlock happen in the next x minutes?" for x between 1 and 10. Figure 1 using our 175,684 samples to visualise our ground truth: did an unlock actually happen in the next x minutes during the duration of the study? As expected, we observe a class imbalance, especially for short time windows (e.g., 1 min had 163129 "no" instances and 12555 "yes" instances). As the time window increase, there are more samples in the class "yes" balancing the two classes (e.g., 10 min has 119831 "no" instances and 55853 "yes" instances).

To better understand the classification performance, we first need to define a baseline classifier. We simply use the statistical distribution of mean times between two sessions (i.e., the idle usage period between two consequitive lock & unlock events). Our baseline classifier outputs positive predictions of the next smartphone unlocking event when the idle usage period is longer than the average value, and vice versa (since the baseline is not an algorithm, it does not give ROC).

Figure 2 depicts the comparison results in overall classification accuracy (i.e., the ratio of correctly classified instances). The baseline classifier's performance is 0.630 regardless of time windows. The accuracy of Random Forests is the highest in all time windows, stably ranging from 0.913 to 0.937.

Figure 3 shows the receiver operating characteristic (ROC) areas of the four classifiers. Here, each classifier is asked to predict whether an unlock

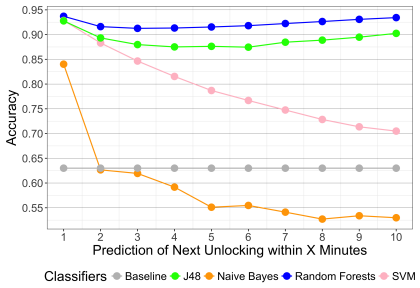


Fig. 2: Comparison between baseline, Naive Bayes, J48 Decision Tree, Random Forests and SVM.

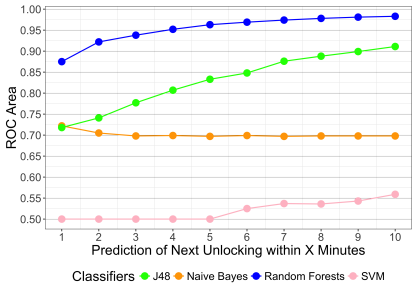


Fig. 3: ROC Areas of Naive Bayes, J48 Decision Tree, Random Forests and SVM.

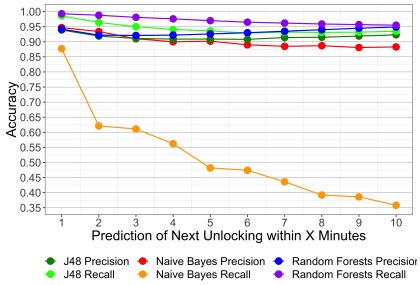


Fig. 4: Precision and recall of Naive Bayes, J48 Decision Tree and Random Forests on class "no"

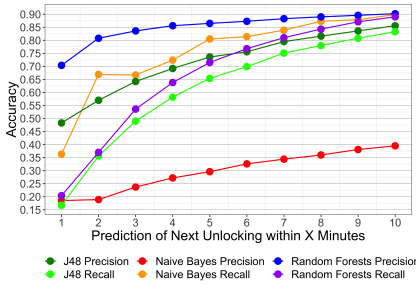


Fig. 5: Precision and recall of Naive Bayes, J48 Decision Tree and Random Forests on class "yes"

event will be observed in the next x minutes (x-axis). Random Forests outperformed other classifiers. Its ROC areas range from 0.875 to 0.983, which show a steady ascending trend as time window increases.

To further investigate the classifiers' performance, we separately consider the two classes: "no" (which represents the cases where the user did not unlock the phone within x minutes), and "yes" (where the user unlocked the phone within x minutes). Figure 4 shows the precision and recall of the three classifiers for the class "no" and Figure 5 for class "yes". In these figures, precision measures the ratio of true positives to all positive predictions on this class. Similarly, recall denotes how many relevant instances are correctly predicted.

For class "no", Random Forests had the highest precision within 0.921 to 0.949. For class "yes" the precision of Random Forests is also the highest among the three classifiers, increasing approximately in a shape of a logarithmic function from 0.704 at 1 min to 0.902 at 10 mins. The recall of Naive Bayes is the highest, significantly outperforming the others from 1 min to 5 mins (growing from 0.363 to 0.899). The difference becomes insignificant in larger time windows.

Since Random Forests had the best and stable performance in our analyses, we only consider this algorithm for further analysis of predicting smartphone unlocking. We actually attempted to apply parameter tuning methods on the RBF kernel for SVM (see Appendix A.1), but due to the large number of possible γ values we could not improve further.

4.2.1 Random Forests Parameter Tuning

Previous work, such as [42], stated that the increase of number of trees in Random Forests can improve the performance, and the default setting of feature number $\lfloor \log_2 m \rfloor + 1$ is optimal (m is the number of initial features). Hence, we attempted to repeat the 10-fold cross-validation on the data of 10 min time window, with different numbers of trees in Random Forests, including increasing and decreasing. As expected, the highest number of trees produced the best performance. The ROC Area grows up from 0.978 at 25 trees to 0.983 at 150 trees; class "no" precision from 0.942 at 25 trees to 0.949 at 150 trees; class "no" recall from 0.952 at 25 trees to 0.955 at 150 trees; class "yes" precision from 0.896 at 25 trees to 0.902 at 150 trees; class "yes" recall from 0.874 at 25 trees to 0.889 at 150 trees.

The results indicate that increasing the number of trees improve the overall performance of Random Forests. The default value of the number of trees in Random Forests is 100, which is already large. Considering limited computing resources on smartphones, it is not necessary to use a larger number, unless the targeted mobile applications have strong need in the high performance of classification.

4.2.2 Optimisation regarding Class Imbalance for Random Forests

From Figure 5, we observed that using a small time window significantly reduced the classification performance of Random Forests for class "yes". This may be because there are fewer data instances within the class "yes" in small time windows, which cause a class imbalance problem [26]. The class imbalance problem represents the challenge in the cases where data instances within one class are significantly more or less than those within other class(es). Algorithms such as Random Forests may perform poorly in predicting the class with an imbalanced number of instances. Similar to previous work, such as [26], handling this problem, we attempted to apply repeated random sub-sampling to improve the performance of Random Forests in small time windows. Repeated random sub-sampling is a popular method for highly imbalanced datasets (e.g., medical records containing a large number of negatives and a several positives) [26].

Figure 6 summarises the process of repeated random sub-sampling for the construction of unlocking prediction models using Random Forests. First, given the training data with N_0 instances of class "no" and N_1 instances of class "yes", it computes NoS, where $NoS = \lfloor \frac{N_0}{N_1} \rfloor$. If $NoS < 3$, it ends the process by assuming that there is no class imbalance problem. Otherwise ($NoS \geq 3$, which means $N_0 \gg N_1$, and at least 3 models can be voters), it randomly divides N_0 instances of class "no" into $N0S$ groups. Then, it trains NoS models where each model is trained by one subgroup of instances from the class "no" and all instances from the class "yes".

Figure 7 summarises the prediction process. The values from all features are sent to all the NoS models. The results from the the NoS models go to a voting system which simply accepts the prediction of the majority. In the case where there are equal numbers of positive and negative predictions from the models, it gives the negative prediction (i.e., class "no"), since the negative instances are the majority in training data.

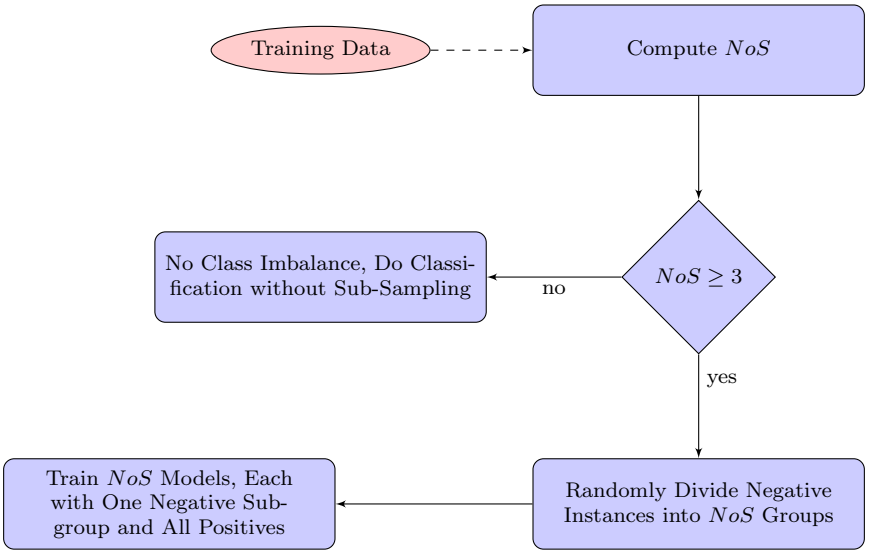


Fig. 6: Training models of Random Forests with sub-sampling against class imbalance. NoS is the ratio of "no" to "yes" instances.

With the repeated random sub-sampling method, we repeated the 10-fold cross-validation on the data of all time windows using Random Forests. The data sets of time windows from 1 min to 7 mins were considered as imbalanced ($NoS \geq 3$).

Figure 8 compares the process with and without repeated random sub-sampling using the precision and recall measure for the class "no". And Figure 9 shows the results for the class "yes". Overall, repeated random sub-sampling significantly improved the performance for the class "yes" which was challenged by the class imbalance problem. However, repeated random sub-sampling reduced the performance for the class "no" which had the majority of instances. For various mobile applications, the two classes may be of different importance. Developers should use repeated random sub-sampling if their applications need higher performance of unlocking prediction for the class "yes" using small time windows.

4.3 Feature Evaluation in Classification

To quantify the importance of each feature, we compared them using two criteria. First, we used the "select attributes" function of Weka to generate the average ranking of features in 10-fold cross validation. The evaluator used was the classifier *attributes subset* which measures the accuracy decrease caused by the removal of features from a classifier. Since Random Forests performed the best in the previous analyses, we only consider this classifier. The search method was *Greedy Stepwise* which explores forward or backward in a greedy search through space within feature subsets. Second, we also included two feature importance measures generated by the training process of Random Forests in 10-fold cross validation: Mean Decrease Accuracy (MDA) and Mean Decrease Impurity (MDI).

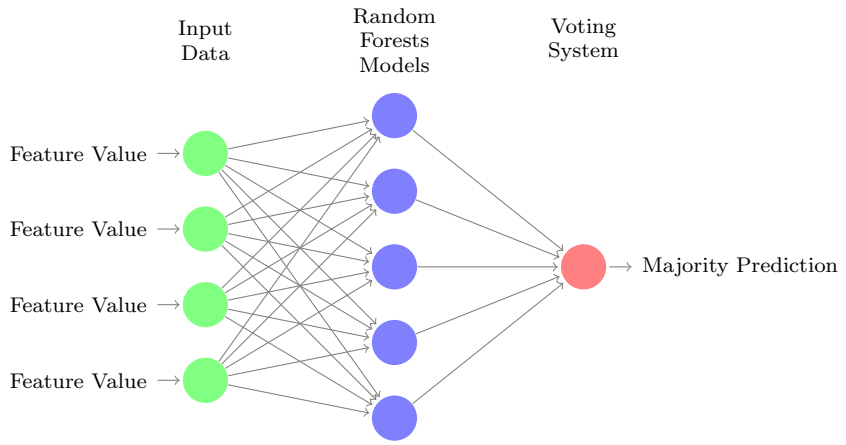


Fig. 7: Using models of Random Forests with sub-sampling to classify samples with class imbalance

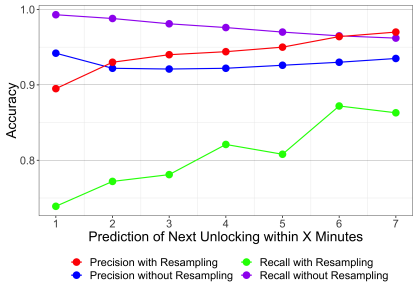


Fig. 8: Precision and recall of Random Forests with and without sub-sampling on class "no"

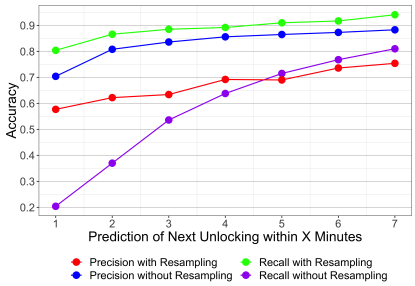


Fig. 9: Precision and recall of Random Forests with and without sub-sampling on class "yes"

Since Random Forests gained the highest ROC area in the time window 10 mins, we used this timeframe to maximise the effect of using different feature subsets (also, using this timeframe can minimise the effect of class imbalance which is illustrated in the discussion section).

Table 8 includes the importance of all the 19 features computed using our two methods. According to Weka's "select attributes", the 5 features with the highest average rank are: time since the end of last session, duration of last session, hour of day, activity, and minute of hour. Among them, only activity data is collected from hardware sensors. The 5 features with the highest MDA on the class "no" are: time since the end of last session, WiFi data upload, WiFi data download, duration of last session, hour of day. Only 2 of them (WiFi data upload, WiFi data download) are hardware-related features. Similarly, the 5 features with the highest MDA on the class "yes" are: time since the end of last session, WiFi data upload, WiFi data download, hour of day, number of foreground apps used in last session. Still, 3 of them are software-related features.

Table 8: For the binary classification problem, the importance of all the features calculated by select attributes (average rank) and the two kinds of measures (MDA and MDI) of Random Forests.

Feature	Rank	No-MDA	Yes-MDA	All-MDA	MDI (Gini)
Time since last session	1	0.16681	0.23758	0.18931	11982.5
Duration of last session	2	0.11121	0.11011	0.11086	6669.2
Hour of day	4.3	0.11097	0.12337	0.11491	4931.3
Activity	4.4	0.05428	0.03985	0.04969	2092.5
Minute of hour	5.5	0.03897	0.07236	0.04958	5105.0
Day of week	5.9	0.05578	0.06556	0.05889	2999.3
Proximity	7.5	0.03234	0.02655	0.03050	882.9
Ambient light	8.1	0.08299	0.09787	0.08772	3719.0
Cellular data download	8.8	0.08347	0.08378	0.08357	4173.8
Cellular data upload	10.1	0.08838	0.08568	0.08753	4136.5
WiFi data download	10.3	0.13248	0.12501	0.13010	5094.3
WiFi data upload	12.1	0.13656	0.12556	0.13306	5067.1
Step count since last session	12.3	0.06695	0.07625	0.06991	2776.5
Number of foreground apps used in last session	14.1	0.10863	0.11551	0.11082	3274.9
GPS availability	15.3	0.03025	0.03679	0.03233	773.4
WiFi spots nearby	16.2	0.03636	0.04596	0.03941	2025.4
Network location availability	16.5	0.02753	0.03404	0.02960	752.3
WiFi availability	16.6	0.01905	0.02253	0.02015	524.7
Gender	19	0.06928	0.09512	0.07749	1179.9

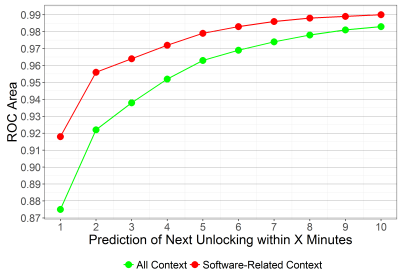


Fig. 10: ROC areas of Random Forests using all context and software-related context.

4.4 Classification using only Software-Related Context

To reduce power consumption in prediction, we calculate the prediction accuracy without considering context derived from hardware sensors. Thus, we only selected the following "software-related" features: application usage, screen events, time and gender.

Figure 10 shows the ROC areas of Random Forests using all context and software-related context. Using software-related context only, Random Forests actually achieve a slightly higher ROC area curve (from 0.918 at 1 min to 0.990 at 10 mins) than using all available context (from 0.875 at 1 min to 0.983 at 10 mins). As the time window grows, the performance gap gradually decreases. Figure 11 and 12 show the precision and recall of Random Forests using all context and software-related context for both classes "no" and "yes". Precision and recall actually improves when using software-related context.

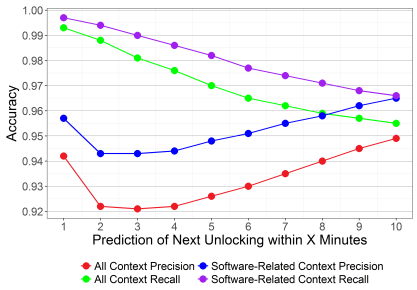


Fig. 11: Class "no": precision and recall of Random Forests using software-related context vs. using all context

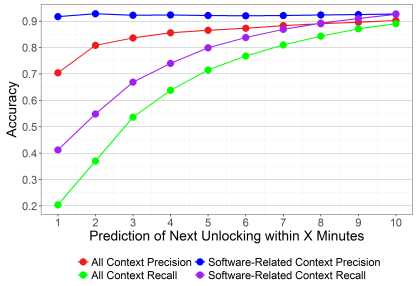


Fig. 12: class "yes": precision and recall of Random Forests using software-related context vs. using all context

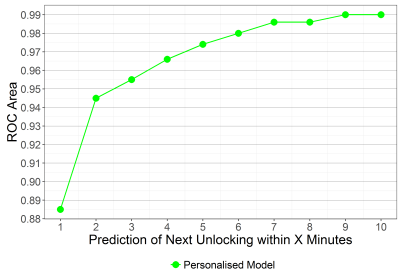


Fig. 13: Using software-related context only, ROC area of the personalised model trained by Random Forests.

4.5 Personalised Model in Classification

To validate our approach under more realistic conditions, we investigated personalised models built by data from each participant. Thus, we personalised the model for each participant using Random Forests and software-related context. We conducted 10-fold cross validation on each participant's own data. Then we computed the average ROC areas across all participants.

Figure 13 depicts the ROC areas of the personalised models. The ROC area of the personalised model grows rapidly as the time window increases from 1 min to 5 mins. For longer time windows, the ROC areas plateau.

Figure 14 shows the precision and recall of the personalised model for the class "no". The precision curve of the personalised model is stably high, between 0.958 and 0.970, in all time windows. In contrast, the recall curve, although on a higher level, follows a descending pattern: reaching the peak on 0.993 in time window at 1 min, then steadily dropping at a slow speed as the time window grows (down to 0.981 at 10 mins).

Figure 15 compares the precision and recall of the personalised model for the class "yes". The precision curve grows steadily from 0.854 to 0.959 in time window from 1 min to 10 mins. Comparatively, the recall curve grow rapidly from the level below 0.5: from 0.421 at 1 min to 0.695 at 3 mins. Then it grows gradually in larger window sizes (up to 0.904 at 10 mins).

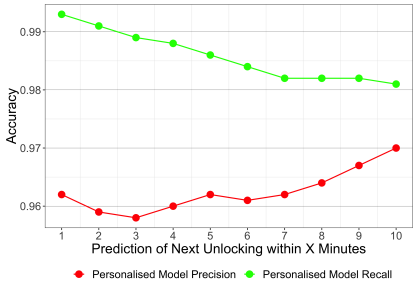


Fig. 14: Class "no": precision and recall of the personalised model trained by Random Forests

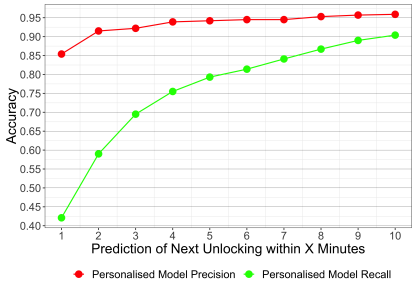


Fig. 15: class "yes": precision and recall of the personalised model trained by Random Forests

4.6 The New User Cold Start Problem

A common scenario for classification system is the cold-start problem, where we need to make inferences or predictions about a new user who has not provided any information. In this case, our classifier needs to use data collected from other users to make a recommendations for new users. This challenge is called "the new user cold start problem" [6]. In the same manner, our system predicting the next smartphone unlocking event also has to handle this problem. A new user may want to use this system immediately after installation, meaning that there will be no training data from this user for the system to learn and build a personalised model. The system must generate predictions using the model trained by data collected from previous users.

Consequently, we investigated how a cold start of smartphone unlocking prediction performs on a new user. Similar to previous work [6], we conducted a leave-one-out cross validation which iterates through each user by considering the data from other users as the training data, and then validating the model using the data of this user as the testing data. The evaluation conditions are the same as the analysis of personalised models: using Random Forests and software-related context.

Figure 16 compares the ROC areas of the model in cold start and the personalised model. Across all time windows, the ROC areas of the model in cold start remains low, ranging from 0.583 to 0.594. In contrast, the ROC areas of the personalised model are significantly higher, growing up steadily as the time window increases.

Figure 17 shows the precision and recall of the model in cold start and the personalised model for the class no. As the time window increases, both the precision and recall curve of the model in cold start drop steadily: precision from 0.936 at 1 min to 0.732 at 10 mins; recall from 0.994 at 1 min to 0.788 at 10 mins. Comparatively, the curves of the personalised model are stably high.

Figure 18 depicts the precision and recall of the model in cold start and the personalised model for the class yes. As the time window increases, both the precision and recall curve of the model in cold start follow a slowly ascending pattern: precision from 0.256 at 1 min to 0.375 at 10 mins; recall from 0.004 at 1 min to 0.296 at 10 mins. The curves of the personalised model also follow an ascending pattern with significantly higher values in different time windows.

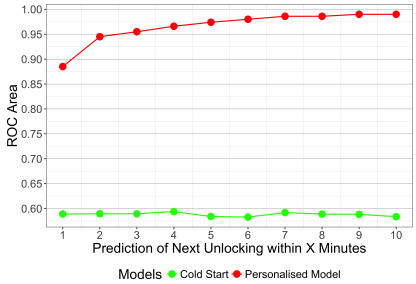


Fig. 16: ROC Areas for our classifier in cold start vs. personalised model.

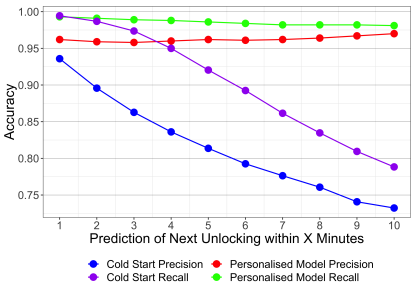


Fig. 17: Class "no": precision and recall of the model in cold start, compared to the personalised model

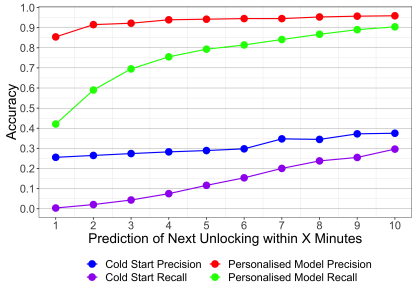


Fig. 18: class "yes": precision and recall of the model in cold start, compared to the personalised model

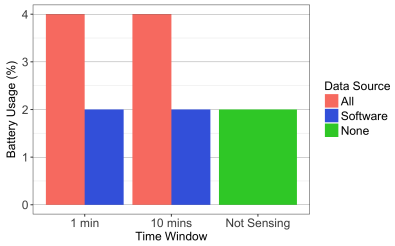


Fig. 19: Energy consumption of our approach on Huawei GR5 2017 in different conditions.

4.7 Energy Consumption

As energy efficiency is a critical requirement of our approach, we quantified the energy consumption on a real smartphone in different conditions. We selected a Huawei GR5 2017 with Android 7.0 as the experiment device, with 3340mAh full battery capacity.

We installed AWARE and our implementation of the unlocking prediction system which is based on the Random Forests classifier in Weka. We compared two sets of contextual data sources (all data sources vs. software-related sources) and two time windows (1 min vs. 10 mins). Also, to provide a baseline, we measured the energy consumption in standby state by disabling the installed applications on the device. The test for each condition ran for 24 hours.

Figure 19 shows the results of the energy tests in different conditions. When using all the data sources, the device used 4% (133.6mAh) of its battery capacity when using a prediction with time window either 1 min or 10 mins. In contrast, the devices used only 2% (66.8mAh) battery when using software-related data sources only for time windows either 1 min or 10 mins. This energy consumption is virtually the same as that of standby state, meaning that using software-related data sources only has negligible impact on smartphone battery.

5 Discussion

Our work has shown that it is possible to predict whether users will unlock their phone in the near future. Our analysis shows that contextual data can be used to make this prediction with varying degrees of confidence. We adopted two independent methods to make this prediction: using regression, and using classification. Classification allows us to ask whether the user is likely to unlock their phone in the near future (up to 10 minutes). This approach is suitable when a short-term prediction is required. Regression enables us to predict the timing when we expect the next unlock to happen. This approach can be useful for longer-term planning. For example, if the classification suggests that the user will not unlock their phone in the next 10 minutes, regression can be used to estimate when the next unlock may take place.

Additionally, our work has investigated the use of software-only sensors, as a means of reducing the power consumptions necessary for prediction. The analysis shows that in fact using software-only sensors improves predictive power. Finally, our analysis shows that personalised models perform quite well and on par with general models, but in cold-start settings the performance significantly drops.

5.1 Regression Methods

We investigated the effectiveness of regression methods attempting to predict the exact time of the next unlocking event. Our findings show that Random Forests can perform significantly better than multiple linear regression. This indicates that the relation between smartphone usage patterns and phone unlock events are highly nonlinear. We also found that the difference between using all features and using software-related features only was insignificant in this case. Further, a software-related feature, time since last session, had the strongest correlation to the time before the next unlocking event. Using personalised models and software-related features, Random Forests with the regression model can achieve an MAE of 8.89 mins and an RMSE of 24.44 mins. This accuracy is sufficient to schedule computation-intensive tasks, such as OS updates taking half an hour, when the classifier predicts a very long period of idle phone usage.

The strength of regression methods is flexibility, since algorithms can estimate the time when users are likely to unlock their phone next. One exciting possibility for this kind of prediction is the hypothesis that our classifier can be used to predict when users are going to sleep or will wake up. For example,

previous work at UbiComp has shown that smartphone usage can be used to detect sleep behaviour [1] with MAE 45 minutes.

That work used screen on-off patterns to build personalized models for retrospectively determining sleep. We believe that our regression model can be used to identify long periods when users are not expected to unlock their phone, therefore identifying times when the users may be sleeping. By its nature, our prediction also identifies the moment in time when users are expected to wake up (and start using their phone), as shown in previous work.

Furthermore, mobile applications can use both classifiers and regression methods to improve user experience, particularly for frequent users [45,39]. When all time windows do not fit the requirement of a specific task (e.g., prediction if there will be 30 mins of idle usage), the smartphone can rely on regression. However, we found that the results generated by a cold start of our regression models tend to be very inaccurate on average. Hence, we suggest avoiding regression methods to launch a cold start prediction of unlocking. For a regression-based cold start of unlocking prediction, future research can investigate the possible strategies to improve the performance.

5.2 Classification Methods

Our study with 27 participants demonstrates that it is possible and practical for smartphones to predict the next unlock event using a machine learning model and contextual data. Given the smartphone unlock time prediction as a binary classification problem, results show that a personalised model trained by Random Forests using software-related contextual data achieves ROC areas from 0.885 at 1 min to 0.99 at 10 mins. For every instance where the user will not unlock the phone in the next x minutes (labelled as class "no" in our classification), the prediction accuracy of this model ranges from 0.993 at 1 min to 0.904 at 10 mins. Energy-saving mechanisms such as pausing the downloading of timely content and hardware sensor usage can be triggered by our model's predictions that fall in the class "no". With such mechanisms triggered, the amount of saved energy is the same as the prediction accuracy. When the user is going to unlock the phone in next x minutes (labelled as class "yes"), the performance varies greatly from 0.421 at 1 min (without sub-sampling) to 0.981 at 10 mins. Following a prediction in the class "yes", context-aware services on smartphones may launch in the background to generate relevant content for the next usage session after the expected unlock event.

By empirically validating the predictability of unlock events, our work generalises the modeling of smartphone usage patterns to also include unlocking itself, in addition to existing approaches that predict which app users will launch next [25]. Similar to previous approaches using Random Forests on smartphone such as call-availability prediction [35], boredom detection [36], and gesture recognition [40], our approach is feasible as a stand-alone application for off-the-shelf smartphones.

5.3 Software-Related Context

Based on the results of feature evaluation on the binary classification problem using Weka's "select attributes", and the two importance measures of Random Forests, we found that the strongest indicators of unlock events are software-related features: idle time (time since the end of last session), the duration of last session, and the hour of day. The same holds true for regression: using IncMSE and IncNodePurity we also observed that software-related features lead to better higher accuracy and lower dispersion. For regression, those feature were; time since the end of last session, the duration of last session, and the hour of day. Additionally, we found a strong correlation ($r_s = 0.411$) between idle time (time since the end of last session) and the time before next unlocking event. This correlation is the strongest compared to that of any other feature which is a continuous variable.

This finding is in line with results from van Berkel et al. [5], who compared multiple features to predict whether a user would continue with a previous objective or start a new objective unrelated to the previous task when unlocking their phone. Following from these results, we argue that mobile phone usage behaviour follows a pattern that is related to both the previous user interaction and the time of day. These attributes can be collected by relying solely on context information obtained through the device's software. This means that our approach can run on the personal device of the user without need for hardware sensors or communication with the outside world.

Furthermore, it is interesting to note that Random Forests using software-related context obtains slightly higher prediction accuracy than using all the context. This phenomenon indicates that some features obtained from hardware are correlated with software-related features, or are irrelevant to the unlock events. Redundant features can decrease the classification performance of Random Forests. Also, feature evaluation shows that several hardware-related features such as WiFi availability and GPS availability have significantly low importance, yet are energy intensive.

5.3.1 Energy Efficiency in Unlocking Prediction

Since one of our objectives is to reduce energy consumption in the locked state, an important condition of a successful implementation is that the model itself is energy efficient. In the usage of unlocking prediction, the algorithm works with a trained model on the smartphone, consuming no phone power to train a new model. Specifically, based on our experimental results , we found that using software-related context (i.e., the data collection for screen events and clock time) has negligible impact on smartphone battery. This find is in line with prior work stating that quick computations by CPU and RAM consume only little power in the locked state [9], [17]. As stated in [17], operating systems has optimal mechanisms for system-level broadcasts such as screen and application events. Fetching these events does not involve extra overhead.

Overall, our findings suggest that using only software-related features is a feasible strategy to achieve better accuracy and energy efficiency in performing smartphone unlock prediction, compared to using both software-related and hardware-related features.

5.4 Deployment of Unlocking Prediction: Personalised Model and Cold Start

In practice, systems based on supervised learning have two ways of deployment for a new user: personalising a model using data collected from the new user, or launching a cold start with a model trained by data collected from previous users' data.

Personalised models are commonly used by most context-aware techniques on smartphone. These models gradually learn and adapt to their user. Using personalised models can maximise the accuracy of predictions, as demonstrated in our results 16. When using short time windows, personalised models tend to generate wrong prediction for the class "yes" instances. Repeated random sub-sampling can be used to overcome this problem. Using long time windows, personalised models can generate accurate predictions for both classes.

An important strength of personalised models is the protection of privacy. Since a smartphone collects and processes data from its own user, personalised models do not involve any connection to a network and/or other devices. After training a model, the smartphone can delete the raw data to minimise risks. However, personalised models require that some data collection occurs before they can be used. Although unlocking prediction is an opportunistic sensing system (i.e., users do not explicitly input information to train a model), it may take long time (e.g., 2 weeks in our study) to train a model with sufficient performance. Also, many factors will affect the training cost (time and power usage), such as hardware, data size, algorithm parameters and implementation of software. Since our approach only requires Random Forests algorithm, it can work on any platform including Android and iOS.

In contrast, a cold start can directly install the system with a trained model on the smartphone of the new user, so that the smartphone can have predictions immediately after the installation. This model is trained by data collected from previous users. Although there is no training effort for the new user in a cold start, our findings indicate that this approach has poor performance. Our evaluation results indicate that the models of cold start cannot accurately predict instances of class "yes" regardless of time window sizes. This finding reflects the significant difference of phone usage behaviours across different users. Also, cold start models cannot accurately predict instances of class "no" in long time windows. This means that a cold start may be useful in very few cases.

Also, a cold start implies that previous users have to upload their raw data to a central server that trains the model. It is challenging to recruit such users without sufficient compensation. Even with considerable compensation, this data collection may involve privacy risks that may be harmful to these users and may decrease their willingness of data uploading. In practice, this can be achieved through volunteers that sign up for beta testing.

5.5 Limitations and Future Work

In this study we do not distinguish unlock events caused by notifications or phone calls which users unlock the phone in a passive way (i.e., phone usage without a planned user intention) . This means that future research can investigate smartphone unlock events on a lower level comprising:

- active unlock events initiated by the phone user;
- passive unlock events where the user is notified via sound, vibration, LED light or the bright screen by the phone.

Then the classifier can make more detailed predictions to support certain applications. However, the classification performance on the lower level might decrease for brief time windows because the classes are more imbalanced (active/passive unlock events are subclasses of unlock events.) , as we have discussed in this paper. Based on the predictions of active/passive unlock events, smartphone apps can further benefit users. On the other hand, splitting more classes within unlock events increases the cost of when the model is trained on hardware.

Constrained by our computing resources for data analysis, another limitation is that we did not include diverse demographic features in our model. We had 27 participants (23 with academic background) which cannot capture rich demographic diversity among billions of smartphone users. Many demographic features including age, occupation, education and personality may be useful for the classifier to achieve higher accuracy. These features are also critical to implement an accuracy model for new users in a cold start scenario. Future research can explore possible solutions to improve the model for a cold start. Moreover, changes in demographics may greatly impact phone usage. For example, a user may radically change their phone usage behaviours after moving to another country, so that a model trained in the previous country may have lower performance. Similarly, users' mental states (e.g., depression and anxiety) may be helpful to increase the accuracy the prediction. Developers can connect medical apps as a data source to extend our system. To investigate the effectiveness of additional features, future research can extend our study to a larger group of participants. However, the computational cost (e.g., time and hardware) in data analysis will also be larger. Researchers should have sufficient computing resources such as servers or workstations with large RAM.

Additionally, in the future, researchers can also attempt to generalise our findings to other mobile devices such as tablets and smartwatches. The usage scenarios of our approach on smartphones are similar to those of tablets and smartwatches. If our approach is suitable for these devices, accurate unlock prediction will also be a great benefit for tablet and smartwatch applications.

6 Conclusion

In this paper we propose multiple approaches for using context data to predict when a smartphone will be unlocked. Based on the results of our field study with 27 participants, we found that it is possible to predict the next phone unlock event, using Random Forests as a classifier/regressor and smartphone contextual data including hardware sensor data and phone usage patterns. Furthermore, our feature evaluation indicates that the strongest indicators of phone unlocking are software-related features including idle time (time since the end of last session), the duration of last session, and the hour of day. We also found that using software-related features only can slightly improve the accuracy and enable energy efficiency of continuous sensing in unlocking prediction.

In the comparison between personalised models and cold start, our results show that personalised models can generate better predictions (ROC Area from 0.885 to 0.99 using different time windows), whereas cold start models cannot achieve the same performance. We also show that there exists a trade-off in the time window selection. If the classifier uses a longer time window, the model will have higher accuracy in positive predictions and lower accuracy in negative predictions. To achieve high accuracy in positive predictions by Random Forests using short time windows, we found that the repeated random sub-sampling method is very effective.

In contrast, personalised models using Random Forests in the regression mode can generate the exact time before next unlocking event (MAE = 8.89 mins, RMSE = 24.44 mins), without considering time windows. Our findings enable smartphones applications to collect sensor data or prepare timely content to improve the context-awareness for the next phone usage session. Also, by inferring the next period of the idle state, smartphones applications and operating systems can reduce unnecessary operations to improve energy efficiency, and schedule computation-intensive tasks, such as OS updates, in the locked state to avoid the disturbance of phone usage.

A Appendix

A.1 Parameter Tuning

To improve the performance of SVM, we first tried the linear kernel. However, the performance of SVM with linear kernel degraded ($ROC\ Area = 0.523$ at 10 min) compared to the RBF kernel ($ROC\ Area = 0.571$ at 10 min). Hence, we attempted to apply parameter tuning methods on the RBF kernel. As illustrated in previous work investigating parameter tuning for SVM with the Radial Basis Function kernel [24,10] , we repeated the 10-fold cross-validation on the data of 10 min time window with replacing the default setting $\gamma = \frac{1}{m+1}$ with a wide variety of γ values.

Figure 20 shows the performance of SVM with the RBF kernel having different γ values. Among all γ assignments, $\gamma = 100$ achieved the best performance: $ROC\ Area = 0.813$, class "no" precision 0.864, class "no" recall 0.945, class "yes" precision 0.852, class "yes" recall 0.682.

The results indicate that, given a suitable γ, SVM with the RBF kernel can also achieve high performance in the classification to predict unlocking events. However, due to the large number of possible γ values and our limited computing resources, we could not refine our finding, since parameter tuning is highly time-consuming and computation-intensive (in our case, running each γ value took about 10 days for a normal PC). With $\gamma = 100$, although SVM with the RBF kernel achieved considerably good results, the performance of Random Forests was still better. Hence, we focused on Random Forests in further analysis. Future work may conduct deeper investigation about the employment of SVM with the RBF kernel in phone unlocking prediction.

References

1. Abdullah, S., Matthews, M., Murnane, E.L., Gay, G., Choudhury, T.: Towards circadian computing: "early to bed and early to rise" makes some of us unhealthy and sleep deprived. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '14, pp. 673–684. ACM, New York, NY, USA (2014). DOI 10.1145/2632048.2632100. URL http://doi.acm.org/10.1145/2632048.2632100
2. Avrahami, D., Hudson, S.E.: Responsiveness in instant messaging: predictive models supporting inter-personal communication. In: Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 731–740. ACM (2006)

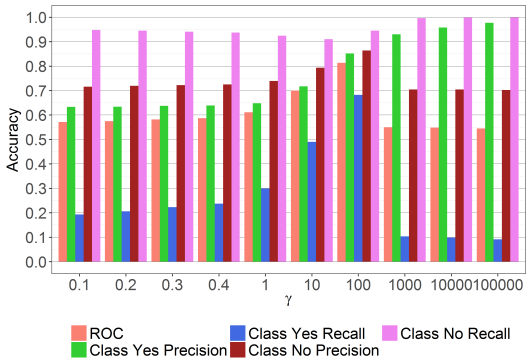


Fig. 20: SVM with the Radial Basis Function kernel

3. Banovic, N., Brant, C., Mankoff, J., Dey, A.: Proactivetasks: the short of mobile device use sessions. In: Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services, pp. 243–252. ACM (2014)
4. Ben Abdesslem, F., Phillips, A., Henderson, T.: Less is more: energy-efficient mobile sensing with senseless. In: Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds, pp. 61–62. ACM (2009)
5. van Berkel, N., Luo, C., Anagnostopoulos, T., Ferreira, D., Goncalves, J., Hosio, S., Kostakos, V.: A systematic assessment of smartphone usage gaps. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, pp. 4711–4721. ACM (2016)
6. Bobadilla, J., Ortega, F., Hernando, A., Bernal, J.: A collaborative filtering approach to mitigate the new user cold start problem. Knowledge-Based Systems **26**, 225–238 (2012)
7. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
8. Brown, B., McGregor, M., McMillan, D.: 100 days of iphone use: understanding the details of mobile device use. In: Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services, pp. 223–232. ACM (2014)
9. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone (2010)
10. Chang, C.C., Lin, C.J.: Libsvm: a library for support vector machines. ACM transactions on intelligent systems and technology (TIST) **2**(3), 27 (2011)
11. Chon, Y., Talipov, E., Shin, H., Cha, H.: Mobility prediction-based smartphone energy optimization for everyday location monitoring. In: Proceedings of the 9th ACM conference on embedded networked sensor systems, pp. 82–95. ACM (2011)
12. Do, T.M.T., Blom, J., Gatica-Perez, D.: Smartphone usage in the wild: a large-scale analysis of applications and context. In: Proceedings of the 13th international conference on multimodal interfaces, pp. 353–360. ACM (2011)
13. Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., Estrin, D.: Diversity in smartphone usage. In: Proceedings of the 8th international conference on Mobile systems, applications, and services, pp. 179–194. ACM (2010)
14. Ferreira, D., Dey, A., Kostakos, V.: Understanding human-smartphone concerns: a study of battery life. Pervasive computing pp. 19–33 (2011)
15. Ferreira, D., Ferreira, E., Goncalves, J., Kostakos, V., Dey, A.K.: Revisiting human-battery interaction with an interactive battery interface. In: Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing, pp. 563–572. ACM (2013)
16. Ferreira, D., Goncalves, J., Kostakos, V., Barkhuus, L., Dey, A.K.: Contextual experience sampling of mobile application micro-usage. In: Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services, pp. 91–100. ACM (2014)
17. Ferreira, D., Kostakos, V., Dey, A.K.: Aware: mobile context instrumentation framework. Frontiers in ICT **2**, 6 (2015)
18. Fischer, J.E., Greenhalgh, C., Benford, S.: Investigating episodes of mobile phone activity as indicators of opportune moments to deliver notifications. In: Proceedings

of the 13th international conference on human computer interaction with mobile devices and services, pp. 181–190. ACM (2011)
19. Genuer, R., Poggi, J.M., Tuleau-Malot, C.: Variable selection using random forests. Pattern Recognition Letters **31**(14), 2225–2236 (2010)
20. Google: Activity recognition api. `https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionApi` (2017)
21. Google: Sensor. `http://developer.android.com/reference/android/hardware/Sensor.html` (2017)
22. Ha, J.H., Chin, B., Park, D.H., Ryu, S.H., Yu, J.: Characteristics of excessive cellular phone use in korean adolescents. CyberPsychology & Behavior **11**(6), 783–784 (2008)
23. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD explorations newsletter **11**(1), 10–18 (2009)
24. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. Machine learning: ECML-98 pp. 137–142 (1998)
25. Jones, S.L., Ferreira, D., Hosio, S., Goncalves, J., Kostakos, V.: Revisitation analysis of smartphone app use. In: International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp, pp. 1197–1208 (2015). DOI 10.1145/2750858.2807542. URL `http://people.eng.unimelb.edu.au/vkostakos/files/papers/ubicomp15.pdf`
26. Khalilia, M., Chakraborty, S., Popescu, M.: Predicting disease risks from highly imbalanced data using random forest. BMC medical informatics and decision making **11**(1), 51 (2011)
27. Khoshgoftaar, T.M., Golawala, M., Van Hulse, J.: An empirical study of learning from imbalanced data using random forest. In: Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on, vol. 2, pp. 310–317. IEEE (2007)
28. Kostakos, V., Musolesi, M.: Avoiding pitfalls when using machine learning in hci studies. interactions **24**(4), 34–37 (2017)
29. Krstic, I.: Behind the scenes of ios security. Black Hat (2016)
30. Lane, N.D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., Campbell, A.T.: A survey of mobile phone sensing. IEEE Communications magazine **48**(9) (2010)
31. Lee, M., Bak, C., Lee, J.W.: A prediction and auto-execution system of smartphone application services based on user context-awareness. Journal of Systems Architecture **60**(8), 702–710 (2014)
32. Liu, Y., Xu, C., Cheung, S.C.: Where has my battery gone? finding sensor related energy black holes in smartphone applications. In: Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on, pp. 2–10. IEEE (2013)
33. Lu, H., Yang, J., Liu, Z., Lane, N.D., Choudhury, T., Campbell, A.T.: The jigsaw continuous sensing engine for mobile phone applications. In: Proceedings of the 8th ACM conference on embedded networked sensor systems, pp. 71–84. ACM (2010)
34. Oulasvirta, A., Rattenbury, T., Ma, L., Raita, E.: Habits make smartphone use more pervasive. Personal and Ubiquitous Computing **16**, 105–114 (2012)
35. Pielot, M.: Large-scale evaluation of call-availability prediction. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 933–937. ACM (2014)
36. Pielot, M., Dingler, T., Pedro, J.S., Oliver, N.: When attention is not scarce-detecting boredom from mobile phone usage. In: Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing, pp. 825–836. ACM (2015)
37. Pielot, M., de Oliveira, R., Kwak, H., Oliver, N.: Didn't you see my message?: predicting attentiveness to mobile instant messages. In: Proceedings of the 32nd annual ACM conference on Human factors in computing systems, pp. 3319–3328. ACM (2014)
38. Poppinga, B., Heuten, W., Boll, S.: Sensor-based identification of opportune moments for triggering notifications. IEEE Pervasive Computing **13**(1), 22–29 (2014)
39. Shin, C., Hong, J.H., Dey, A.K.: Understanding and prediction of mobile application usage for smart phones. In: Proceedings of the 2012 ACM Conference on Ubiquitous Computing, pp. 173–182. ACM (2012)
40. Song, J., Sörös, G., Pece, F., Fanello, S.R., Izadi, S., Keskin, C., Hilliges, O.: In-air gestures around unmodified mobile devices. In: Proceedings of the 27th annual ACM symposium on User interface software and technology, pp. 319–329. ACM (2014)

41. Strobl, C., Boulesteix, A.L., Zeileis, A., Hothorn, T.: Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC bioinformatics **8**(1), 25 (2007)
42. Svetnik, V., Liaw, A., Tong, C., Culberson, J.C., Sheridan, R.P., Feuston, B.P.: Random forest: a classification and regression tool for compound classification and qsar modeling. Journal of chemical information and computer sciences **43**(6), 1947–1958 (2003)
43. Truong, K.N., Shihipar, T., Wigdor, D.J.: Slide to x: unlocking the potential of smartphone unlocking. In: Proceedings of the 32nd annual ACM conference on Human factors in computing systems, pp. 3635–3644. ACM (2014)
44. Verkasalo, H.: Contextual patterns in mobile service usage. Personal and Ubiquitous Computing **13**(5), 331–342 (2009)
45. Xu, Y., Lin, M., Lu, H., Cardone, G., Lane, N., Chen, Z., Campbell, A., Choudhury, T.: Preference, context and communities: a multi-faceted approach to predicting smartphone app usage patterns. In: Proceedings of the 2013 International Symposium on Wearable Computers, pp. 69–76. ACM (2013)
46. Yan, T., Chu, D., Ganesan, D., Kansal, A., Liu, J.: Fast app launching for mobile devices using predictive user context. In: Proceedings of the 10th international conference on Mobile systems, applications, and services, pp. 113–126. ACM (2012)
47. Yuan, Y., Raubal, M., Liu, Y.: Correlating mobile phone usage and travel behavior–a case study of harbin, china. Computers, Environment and Urban Systems **36**(2), 118–130 (2012)
48. Zhuang, Z., Kim, K.H., Singh, J.P.: Improving energy efficiency of location sensing on smartphones. In: Proceedings of the 8th international conference on Mobile systems, applications, and services, pp. 315–330. ACM (2010)

Chapter 6

Discussion

In this chapter, we begin by reviewing how our approaches reflect the research questions stated in Section 1.1.1. Next, we discuss the strengths and weaknesses of laboratory testing techniques in the validation of context-driven features of mobile applications. Then, we provide some deliberations about the impacts of laboratory testing techniques on the software development process. Then, we summarise the limitations of our approaches. Finally, we conclude with several potential directions, as well as some suggestions for future research in this domain.

6.1 Reflection from our approaches to research questions

6.1.1 RQ1: How can context-driven mobile applications be efficiently and systematically validated in the laboratory?

In terms of efficiency and systematism, there are several issues in existing laboratory-based approaches for the validation of context-driven mobile applications, as illustrated in Section 2.3.2. First, using existing approaches, testers may find it inconvenient or impossible to simulate multi-dimensional context with heterogeneous sensor data. Also, most existing approaches focus only on the examination of functional properties. Although several testing tools can check non-functional properties, the majority of them cover only one aspect. Moreover, regarding testing phases, the vast majority of existing approaches can only examine implemented software applications. Few approaches support design phase

testing. More support is needed for multi-dimensional context data simulation, higher coverage on the examination of software properties, and low-cost design phase testing.

Thus, in Chapter 3, we presented TestAWARE that aims to facilitate contextual data simulation and multifaceted examination of software properties. TestAWARE supports the replay of sensor, event and audio data with synchronisation. For all data types, TestAWARE allows testers to set a replay speed faster or slower than the real clock. For data reuse and integration, TestAWARE allows testers to import data from online or local sources, as well as to create arbitrary data using data manipulation functions. To check functional properties, testers can use TestAWARE UI to conduct functional testing in a black-box manner without writing scripts. Also, TestAWARE can record application output and assertion results to check low-level details in a white-box manner. For non-functional testing, TestAWARE can measure processing speed of executions at runtime. For sensor-based mobile applications, TestAWARE can estimate power consumption of each sensor on a specific device model. For mobile applications based on classification or regression algorithms, TestAWARE can track the change of machine learning performance over runtime.

Also, in Chapter 3 , we evaluated TestAWARE with a real-world Android application in a case study. The procedures of this study accord with standards in the software engineering community [12, 163] and computer-human interaction community [25]. Hence, the level of validity is sufficient. Experimental results show that the features of TestAWARE can efficiently find errors and locate them in source code to help developers conduct fast software repair. Experimental results also indicate that TestAWARE can efficiently perform multifaceted validation, including functional and non-functional properties, in laboratory settings. Besides, we found that TestAWARE can launch faster data replay on PC-based emulators than on physical mobile devices. Using TestAWARE with PC-based emulators can quickly perform a complete replay of context constructed from longitudinal datasets over a long period (e.g., 6 months).

However, TestAWARE can only operate with implemented software applications on PC-based emulators or physical mobile devices. This implies that developers cannot use TestAWARE to perform validation before they obtain an implemented and executable software application. Hence, to support design phase testing, we presented two other approaches for two specific cases, as we discuss further in Section 6.1.2 and Section 6.1.3.

6.1.2 RQ2: How can the real-time sensing performance of context-driven mobile applications be efficiently and systematically validated in the laboratory?

Previous work provides limited support to validate the real-time sensing performance of context-driven mobile applications. Also, as discussed in Section 6.1.1, few existing approaches aim at design phase testing, although it is possible to validate the performance of sensors on mobile devices without completely implementing the targeted software application.

Hence, in Chapter 4, we presented an approach that aims to efficiently and systematically validate the real-time sensing performance of context-driven mobile applications at the design phase of software development. To provide an understanding of how this approach can be adopted in practice, we used a smartphone sensor-based real-time data hiding method as an exemplar case. This data hiding method aims to enable smartphone applications to encrypt and embed sensitive sensor data or identification codes into other real-time sensor data streams that are not sensitive. Data hiding and cryptography are two major security techniques widely used for communication systems [90, 135]. Some communication systems use a combination of data hiding and cryptography to enhance information security [119, 162].

With the data hiding method, we show the effectiveness of our approach with experiments on physical devices in a laboratory setting. Since the process operates programmatically on devices, there is no need to recruit developers as participants. We demonstrate that, in the laboratory, it is feasible and efficient to quantify the amount of error (i.e., distortion) caused by modification in sensor signals (besides data hiding, applications compressing the sensing data may involve real-time modification [83]). Such distortion quantification requires only the details of the algorithm design, rather than the implemented software application. For integer types, only the number of digits and LSB used for data hiding are needed. For floating types, only the number of digits, the value of the exponent, and LSB used for data hiding are needed. Experimental results show that the estimated distortion is consistent with the actual distortion. We also show that, using a prototype of sensing application in the laboratory, our approach can measure the performance change caused by different sensing frequencies and device models. Compared to a completely implemented mobile application, it is simple to implement a prototype with the invocation of sensors, so that testers

can perform early and low-cost testing at the design phase. For example, on the Android platform, developers can use Java to write a program accessing many types of sensors in about 50 lines of code (e.g., the exemplar code of sensor usage by Android [65]). Likewise, we demonstrate that the performance properties of different sensors can be efficiently validated using a prototype in the laboratory. Specifically, we found that, on Samsung Galaxy S6 edge (OS: Android 5.1.1), the heart rate sensor hardware does not alter its sampling rate even if the source code attempts to change the sensor settings via Android APIs.

However, to measure the processing speed and CPU utilisation of executions in the laboratory, developers have to implement both of the sensor data collection and data processing modules. Based on the implementation, we demonstrate that our approach can measure execution time, payload count (i.e., how many sensor readings can be processed in a given period of time) and the inclusive time percentage of CPU utilisation (i.e., the proportion of a function in the whole period of CPU utilisation by the thread). Note that 100% inclusive time of a function on a thread does not necessarily mean that it occupies CPU all the time, since the thread may be temporarily suspended by CPU.

In summary, we demonstrate that our approach can efficiently and systematically validate the real-time sensing performance of context-driven mobile applications in the laboratory. Especially, our approach is suitable for low-cost design phase testing with only software design and limited implementation efforts.

6.1.3 RQ3: How can the machine learning design of context-driven mobile applications be efficiently and systematically validated in the laboratory?

Prior work provides considerable support for the validation of mobile applications that rely on machine learning algorithms and contextual data. However, the majority of approaches need to operate with implemented applications. For early stages in development, very little is investigated about the validation of various machine learning design choices before implementation. Also, as discussed in Section 6.1.1, there is a lack of approaches and tools focusing on design phase testing, although machine learning design is supposed to be completed at the design stage.

Hence, in Chapter 5, we presented an approach that aims to efficiently and systematically validate the machine learning design of context-driven mobile applications in the laboratory. Specifically, rather than only measuring accuracy, our approach covers a variety of aspects of the machine learning design related to context-driven mobile applications. To build an understanding of how this approach can be employed in practice, we used as an exemplar case a smartphone-based system that relies on machine learning algorithms and contextual data to predict the next unlock event triggered by users. This system serves as a background application to help other smartphone applications and operating systems improve energy efficiency and minimise disturbance to users. For example, if a user is not about to unlock its smartphone soon, the smartphone can cancel unnecessary executions, or launch computation-intensive tasks, such as OS or application updates. Similarly, if a user will unlock its smartphone soon, the smartphone can prepare timely and context-aware content to improve user experience for the next phone usage session.

Based on this exemplar case, we first demonstrate that, in some scenarios, applications can rely on either of the two machine learning tasks: classification and regression. In our case, the prediction can be considered as a straightforward regression problem: given the contextual data, the algorithm should output the timing of when the user will unlock the phone. On the other hand, the prediction can also be considered as a classification problem: given the contextual data, the algorithm should predict whether the user will unlock the phone within the next x minutes ($x > 0$). Thus, to train a classification model, it is necessary to determine the time window length x.

Since the exemplar case is one of daily-life scenarios involving long-term phone usage, ground-truth labels with context data can only be obtained in the real world, rather than in the laboratory. As a result, we performed data collection with real users in a in-the-wild study. It is worth noting that, for most common scenarios such as using accelerometer to predict physical activities (e.g., walking, standing and running), researchers and developers can download relevant datasets from online data repositories. For instance, the UCI machine learning dataset repository [74] serves as a common benchmark for the validation of the predictive performance of machine learning models. With a suitable dataset reflecting the scenario of the targeted mobile application, the validation of machine learning design can generally be carried out in the laboratory.

Based on the obtained data, we show the effectiveness of our approach with analysis in a laboratory setting. Since the analysis operates programmatically on PC, there is no need to recruit developers as participants. Our approach measures the predictive performance of different algorithms for both classification and regression. For regression, we demonstrate that it is worth trying both linear and nonlinear algorithms to predict an unknown relation (i.e., the target function). In our case, we selected multiple linear regression and Random Forests regressor. We found that the predictive performance of Random Forests is significantly better than linear regression. For classification, we compared a variety of algorithms: Naive Bayes, J48 Decision Tree, Random Forests and SVM. We found that there is a large difference in the predictive performance among them. We also found that the influence of hyper-parameters on the predictive performance of SVM is significant. This indicates that the validation of SVM may take more time than algorithms that are less sensitive to parameter changes. In addition, we observed that the time window length x has considerable impacts on the predictive performance.

Regarding features in classification and regression, we demonstrate that our approach can efficiently quantify the importance of each feature using the "select attributes" function of Weka (for classification), and the feature importance measures of Random Forests (for classification and regression). Besides, we found that, in our case, the most useful features are generated by software, rather than hardware sensors or modules. In the comparison between classification models built by software-generated features and all features, results show that the model built by software-generated features can achieve slightly better predictive performance. This finding not only confirms the feasibility of energy-efficient predictions, but also indicates that a simple model using only software-generated features has higher generalisability than a complex model using a large number of hardware-generated and software-generated features. It is worth noting that hardware-based continuous sensing causes significantly high power consumption [47,92], and that users often have to carry additional bulky chargers, such as power packs and solar chargers, to extend the limited battery life of smartphones [46,48,76]. Also, using software-generated features can avoid the risk of privacy disclosure caused by attacks on certain hardware sensors such as GPS and microphone.

As mobile devices are mostly used by unique individuals, our approach compares personalised models and cold start models in classification. We found

that personalised models can achieve significantly better predictive performance than cold start models in our case. As illustrated in previous study [44], there exists substantial individual difference of interaction patterns among smartphone users. However, cold start models may perform well in other scenarios, such as emotion inference [147] and physical activity recognition [20]. Using cold start models in these scenarios can avoid the training efforts (users have to manually provide labels) after installation and simplify the application structure, although the predictive performance may be slightly lower than personalised models.

Similar to the validation of real-time sensing performance of context-driven mobile applications (as discussed in Section 6.1.2), developers have to implement the contextual data collection module and the machine learning algorithm to measure energy consumption. Since data fidelity does not affect the results, our approach quantifies energy consumption by activating contextual data collection and the prediction algorithm on a physical device in the laboratory. Although requiring implementation efforts, the measurement on real devices can accurately reflect energy consumption in practice.

Overall, we demonstrate that our approach can perform efficient and systematic laboratory-based validation for the machine learning design of context-driven mobile applications. Based on an exemplar case, we highlight that our approach can validate most machine learning design choices at the design phase without implementation efforts.

6.2 Strengths and weaknesses of laboratory testing

For most scenarios such as daily smartphone usage, testers can perform laboratory testing on their applications using existing datasets. Laboratory testing simulation tools can also create arbitrary low-level context events, such as a fake GPS location. Using simulated data can avoid cost in participant recruitment for real-world tests. For the validation of machine learning design, if testers cannot find an available dataset, they can first conduct an in-the-wild study with real-world users to obtain a dataset with ground-truth labels. Then they can launch different lab-based tests using the obtained dataset, without repeating real-world tests. Unlike real-world tests with high randomness in context (e.g., traffic conditions of a city may never be the same), lab-based tests can simulate

identical context to reliably reproduce software flaws to help testers find the causes and locate bugs in source code.

In addition, laboratory testing tools can accelerate or slow down the speed of context replay. This feature can help testers quickly validate the long-term behaviours of their applications. For example, PC-based emulators can replay more than 3000 sensor data instances per second, while real-world sensors usually generate only 5-50 readings per second. In practice, testers usually do not have sufficient time to perform real-world longitudinal experiments (e.g., validating diabetes monitoring applications requires 6 months). Note that testers often have to perform multiple rounds of regression testing (i.e., the examination of influences caused by software changes [159]) when new functionality is introduced in a new version.

Similarly, laboratory testing tools are able to perform some special types of testing that cannot be manually conducted by human in the real world, such as stress testing [79, 113]. Since these tools are automated, they can generate a heavy load of events (e.g., launching and stopping applications) over long time.

Also, unlike real-world tests requiring completely implemented software, laboratory testing can be conducted at any stage of software development, as we discuss further in Section 6.3.

Although validating mobile applications in lab-based environments has many benefits, laboratory testing has several weaknesses. First, laboratory testing is not suitable for some types of scenarios. For example, laboratory testing tools can hardly simulate context to validate context-aware control systems, such as camera-based guidance, navigation and control systems on unmanned aerial vehicles [40]. Real-world tests are needed for this kind of scenarios, since the context input depends on both real-time control commands and dynamic environmental factors (e.g., sunlight and wind). Besides, laboratory testing tools often have to modify applications or OS for some kinds of context replay in the black-box testing. Ideally, developers tend to keep applications and OS unchanged in the black-box testing. Managing a modified version of an application or OS imposes a higher technical barrier for testers (e.g., rooting Android OS [38, 148], iOS jailbreak [108]). Developers have to spend time and resources on the modified version. A modification in the targeted application or OS may affect some performance properties, causing inaccurate measurement results. Even if testers do not have

to modify applications or OS, they have to spend time learning to correctly use laboratory testing tools which may be very hard to setup and use.

In summary, for validating context-driven features of mobile applications, laboratory testing can overcome the drawbacks of real-world tests in some conditions where the mobility of users and mobile devices leads to high complexity or cost in the real world. However, laboratory testing cannot yet be considered as a replacement of real-world tests in all cases in the development of mobile applications. It is necessary to consider the usage scenarios, software/hardware dependencies, dataset availability and testers' ability to assess the feasibility of laboratory testing for a certain mobile application.

6.3 Testing is not a phase: the impacts of laboratory testing on the software development process

The concept of "testing is not a phase" was proposed more than 20 years ago [13,14,102], and has been becoming increasingly popular within developers and researchers due to the trend of agile testing [70,146]. Prior work found that software flaws can emerge at any stage, not only in coding activities [77]. Rather than performing testing after implementation, it is a better choice to carry out continuous testing throughout the entire life cycle of software development. Early detection of software flaws leads to lower complexity and cost for developers to repair [118,142]. However, this concept is not widely researched in the development of mobile applications. Few approaches focus on the all-stage validation of context-driven features of mobile applications using laboratory testing.

Thus, in this section we discuss the impacts of laboratory testing on the software development process of context-driven mobile applications. We argue that, beyond existing techniques, laboratory testing as a concept can provide additional support for some testing activities in the requirements, design and program phase of context-driven mobile application development.

6.3.1 Requirements phase testing

Many researchers suggest that developers should begin testing at the requirements phase, before any design or coding is done [81,111,118]. Requirements

phase testing can avoid defects caused by improper requirement analysis. Finding such defects at a later stage requires more time and resources.

In the development of mobile applications with context-driven features, laboratory testing can help developers in multiple aspects of the requirement engineering process specified in international standards [73, 75]. For instance, developers can investigate the feasibility of hardware and software dependencies in requirements. For example, using the data collection middleware AWARE [47] and context data visualisation tools (e.g., ContextViewer [23]), developers can preliminarily understand the characteristics of various sensors on mobile devices. Most of these tools contain a user-friendly interface so that developers do not have to write code for trials. Also, they can verify compatibility between context management middleware and hardware specified in requirements. In addition, for context data storage, developers can investigate and try out the data access modules of middleware and laboratory testing tools to construct feasible database requirements.

6.3.2 Design phase testing

Prior work found that there is a positive correlation between the numbers of software design flaws and software defects [33]. During the design phase, developers often have a wide variety of choices to meet software requirements [118]. Hence, it is important to efficiently examine the feasibility of software design before implementation.

In the development of context-driven mobile applications, previous research shows that the tradeoff among energy consumption, latency and accuracy of machine learning classifiers can be balanced using lab-based simulation at the design stage [30]. In this thesis, we demonstrate that several real-time sensing performance properties and machine learning design choices can also be validated using laboratory testing at the design phase. Beyond these aspects, properties such as scalability can also be examined. For example, a smartphone application may rely on the connection with multiple wearable sensors on a user [26, 115]. At the design phase, laboratory testing techniques can be developed to estimate the bandwidth performance of the connection from a number of wearable sensors to the smartphone application under specific protocols such as Bluetooth.

6.3.3 Program phase testing

Even if the software requirements and design are completely correct, programmers may still make mistakes in implementation. Program phase testing can find not only coding mistakes, but also mistakes caused by wrong or neglected decisions in an early stage [118].

As illustrated in Section 2.3.2, there are diverse approaches and tools for testers to perform program phase testing on implemented context-driven mobile applications in the laboratory. In Chapter 3, we proposed TestAWARE to facilitate efficient and systematic validation of implemented context-driven mobile applications at the program phase. These approaches and tools can help testers examine both functional and non-functional properties. Laboratory testing tools can not only report the existence of flaws, but also reproduce bugs and locate them in source code. Then, testers can inform developers related to the buggy code so that the development team can quickly fix the flaws. Based on the state of the art, future laboratory testing tools are likely to further improve the efficiency and systematism of validation by covering more software properties and providing acceleration for context simulation.

6.4 Limitations

In Chapter 3, we found that the context data replay of TestAWARE may have compatibility issues due to the Android fragmentation problem [68, 116, 156] (i.e., the behaviours of Android OS and hardware may vary among different device models.) This fragmentation problem causes inconsistent sensor value representations on devices made by different manufacturers. For instance, some manufacturers use $\{0, 1\}$ to represent the positive and negative state of proximity sensor. In contrast, other manufacturers adopt $\{0, 15\}$ or $\{0, 100\}$. An application on a mobile device may not correctly recognise the context simulated by data collected on another device. Hence, device-specific transformation of sensor value representations is needed. Also, because AWARE cannot collect UI input from users, TestAWARE does not include UI events into simulation. Although TestAWARE can replay audio streams from WAV files, TestAWARE does not support the replay of video streams for testers to validate camera-based mobile applications. Currently, TestAWARE is implemented on Android and its architecture may not be suitable for iOS which applies a strict policy for inter-process

communication [3]. Besides, when replaying heterogeneous data, the maximal replay speed of each single data type on TestAWARE may be slower than our experimental results if data types are more than the number of CPU cores.

In Chapter 4, we only considered the real-time sensing scenarios using regular sensors. We did not investigate the processing of audio and video data. Most recent mobile processors have specialised components to accelerate audio and video processing. Thus, quantifying CPU utilisation is not sufficient to measure the performance of audio and video processing on these processors. Also, our approach lacks automation support with user-friendly tools and interfaces.

In Chapter 5, we only took into account classification and regression. Our approach cannot be generalised to other machine learning tasks, such as reinforcement learning [150]. Regarding feature importance, we only considered the predictive performance and whether a feature is generated by hardware or software. We did not optimise models by computing other feature acquisition cost (e.g., applications may collect data from web services that involve payment [164]). Also, since rapid advances of hardware technique may provide more useful context information and reduce power consumption in future, combining hardware and software context information can achieve better prediction performance without significant energy cost in the same case . For the execution of our approach, we did not develop user-friendly tools and interfaces to automate the comparison among different design choices.

6.5 Potential directions and suggestions for future research

Although existing laboratory testing techniques can simplify the validation of context-driven features on mobile applications in many scenarios, there are still some challenges that prevent testers from maximising the validation efficiency and systematism. Hence, for future research, we discuss some potential directions and give some suggestions below.

6.5.1 Context simulation

Context simulation is necessary to drive and exercise the context-driven programs of mobile applications. With advances in small-scale electronics, mobile applications developers may need to simulate types of context information from new hardware modules integrated on future mobile devices. A future direction can be the design and implementation of novel context simulators that supports more data types, including UI actions, system events, sensor readings, audio and video streams. Especially, since audio and video frames in multimedia files do not contain explicit timestamps, it is challenging to synchronise, accelerate or slow down the speed of replaying these two streams with other timestamped events and data. Also, considering that mobile devices may be connected to external sensors such as near-infrared spectroscopy (NIRS) [85], future research can develop specialised tools to simulate the readings from these external sensors.

Besides, to provide context-aware services, mobile devices and sensors are increasingly connected with each other via device-to-device communication in the Internet of things [19,130]. Future research can investigate the validation of these services in the laboratory. Also, under the coexistence of wireless networks (e.g., Bluetooth, Wi-Fi and ZigBee [5]), mobile devices have to operate with communication interference which results in a significant increase of packet error rates [112]. For the validation of mobile applications relying on wireless networks, simulation techniques can be developed to simulate the coexistence of these wireless networks and to quantify the interference impacts on the application performance. As mobile applications are designed for various purposes, the involved context features may vary. It is up to developers to decide which context features should be simulated.

6.5.2 Validating real-time sensing

Real-time sensing applications are not limited to single devices or sensors. For real-time sensing applications relying on sensor readings from multiple devices, the inconsistent drifts of device clocks can affect context recognition in the long term [93,98]. Future validation techniques can take into account clock drifts of multiple devices. Also, new techniques can be proposed to validate the real-time sensing performance of microphone-based or camera-based applications. Since audio and video processing algorithms may receive hardware acceleration by

specialised modules on modern mobile processors [78], performance testing techniques may have to obtain workload and memory information from these specialised modules, rather than only monitoring CPU.

6.5.3 Validating machine learning design

To support the validation of machine learning design, new techniques can be proposed to simplify the analysis of predictive performance in classification and regression, such as providing a user-friendly interface to specify settings (e.g., algorithms, grid search of hyper-parameters, time window lengths in classification) and to compare personalised models with the cold start models for some scenarios. Also, new tools can be designed to evaluate the feature importance in machine learning models with respect to multiple factors, such as predictive performance and feature acquisition cost (e.g., power consumption, price of network connections or data access). Beyond classification and regression, future research can investigate how other types of machine learning design, such as reinforcement learning [150], can be validated in the laboratory.

Chapter 7

Conclusion

This thesis explored new techniques that can efficiently and systematically validate context-driven features of mobile applications in the laboratory to avoid or reduce high-cost, time-consuming and impractical real-world tests. The literature shows that existing laboratory testing techniques are effective and can replace real-world tests in some scenarios. However, prior work has several drawbacks. To apply existing techniques, developers often need significant technological resources to set up different testing tools which focus on few aspects of validation. Also, previous work lacks support to perform design phase testing which can expose flaws at the early stage of the mobile application development process. Thus, there is little support for developers to achieve high efficiency and systematism in the validation of context-driven features of mobile applications.

To alleviate these drawbacks, we proposed three approaches in this thesis to answer the following research questions:

- **RQ1. How can context-driven mobile applications be efficiently and systematically validated in the laboratory?**
- **RQ2. How can the real-time sensing performance of context-driven mobile applications be efficiently and systematically validated in the laboratory?**
- **RQ3. How can the machine learning design of context-driven mobile applications be efficiently and systematically validated in the laboratory?**

7.1 Summary of contributions

In Chapter 3, we presented TestAWARE, a laboratory-based testing tool that facilitates efficient and systematic validation of context-driven mobile applications on Android. TestAWARE can help testers validate functional properties and multiple non-functional properties of mobile applications. TestAWARE supports the context simulation of sensor, event and audio data in a synchronised manner. TestAWARE also supports replay speed control that allows testers to specify a multiple of the original speed, so that TestAWARE can accelerate or slow down the simulation to quickly go through longitudinal datasets or to carefully examine complex executions. Besides, TestAWARE can import data from online databases or local sources. For the simulation of uncommon context events, testers can create arbitrary context data using data manipulation functions. We demonstrated that TestAWARE can help testers perform efficient and systematic validation to detect and locate flaws.

In Chapter 4, we focused on the validation of real-time sensing performance of context-driven mobile applications. We showed that the performance properties of real-time sensing applications can be efficiently and systematically measured in the laboratory at the design phase with only software design and limited implementation efforts. We presented an approach that covers multiple aspects of performance validation, including quantifying the amount of error caused by modification in sensor signals, measuring the processing speed and CPU utilisation of executions, measuring the performance change caused by different sensing frequencies, sensor types and device models. With a smartphone sensor-based real-time data hiding method as an exemplar case, we highlighted the effectiveness of our approach.

In Chapter 5, we focused on the validation of machine learning design of context-driven mobile applications. We presented an approach that takes into account a variety of aspects of machine learning design. We demonstrated that, in some scenarios, applications can adopt either of the two machine learning tasks: classification and regression. Based on datasets obtained in the real world, our approach helps testers conduct design phase testing in the laboratory to measure the predictive performance of different algorithms for both classification and regression, to quantify the importance of each feature, to compare models built by software-generated features and all features, to compare personalised models and cold start models in classification, and to quantify energy consumption. We

selected as an exemplar case a smartphone-based system relying on machine learning algorithms and contextual data to predict the next unlock event triggered by users. With the selected exemplar case, we showed the effectiveness of our approach.

7.2 Final remarks

In summary, laboratory testing is a promising method for testers to validate context-driven features of mobile applications with minimal efforts of conducting real-world tests. In the trend of agile methods and "testing is not a phase", laboratory testing is suitable for requirements, design and program phase testing in the life cycle of software development. In this thesis, we proposed new techniques and presented insights to efficiently and systematically validate context-driven features of mobile applications in the laboratory. Currently, laboratory testing still has constraints in terms of usage scenarios, software/hardware dependencies, dataset availability and testers' ability. As advanced software/hardware techniques, user-friendly testing interfaces and open data are being increasingly prevalent, the barrier of adopting laboratory testing in practice is likely to be significantly reduced in the future.

Bibliography

[1] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *International symposium on handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[2] Sharad Agarwal, Ratul Mahajan, Alice Zheng, and Victor Bahl. Diagnosing mobile applications in the wild. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 22. ACM, 2010.

[3] Mohd Shahdi Ahmad, Nur Emyra Musa, Rathidevi Nadarajah, Rosilah Hassan, and Nor Effendy Othman. Comparison between android and ios operating system in terms of security. In *2013 8th International Conference on Information Technology in Asia (CITA)*, pages 1–4. IEEE, 2013.

[4] Wi-Fi Alliance. Wi-fi. `https://www.wi-fi.org/`, April 2019.

[5] Zigbee Alliance. Zigbee. `https://www.zigbee.org/`, April 2019.

[6] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, and Nicola Amatucci. Considering context events in event-based testing of mobile applications. In *2013 IEEE sixth international conference on software testing, verification and validation workshops*, pages 126–133. IEEE, 2013.

[7] Apple. ios. `https://www.apple.com/ios/`, April 2019.

[8] Apple. Xcode. `https://developer.apple.com/xcode/`, April 2019.

[9] AWARE. Sensor. `http://www.awareframework.com/sensors/`, April 2019.

[10] Murat Aydemir and Korhan Cengiz. Emerging infrastructure and technology challenges in 5g wireless networks. In *Computer and Energy Science (SpliTech), 2017 2nd International Multidisciplinary Conference on*, pages 1–5. IEEE, 2017.

[11] Nathaniel Ayewah, David Hovemeyer, J David Morgenthaler, John Penix,

and William Pugh. Using static analysis to find bugs. *IEEE software*, 25(5):22–29, 2008.

[12] Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 272–285. ACM, 2007.

[13] Osman Balci. Principles and techniques of simulation validation, verification, and testing. In *Proceedings of the 27th conference on Winter simulation*, pages 147–154. IEEE Computer Society, 1995.

[14] Osman Balci. Verification, validation, and testing. *Handbook of simulation*, 10:335–393, 1998.

[15] Thomas Ball and Sriram K Rajamani. The s lam project: debugging system software via static analysis. In *ACM SIGPLAN Notices*, volume 37, pages 1–3. ACM, 2002.

[16] Sergi Barrantes, Antonio J Sánchez Egea, Hernán A González Rojas, Maria J Martí, Yaroslau Compta, Francesc Valldeoriola, Ester Simo Mezquita, Eduard Tolosa, and Josep Valls-Solè. Differential diagnosis between parkinson's disease and essential tremor using the smartphone's accelerometer. *PloS one*, 12(8):e0183843, 2017.

[17] Sofia Bekrar, Chaouki Bekrar, Roland Groz, and Laurent Mounier. Finding software vulnerabilities by smart fuzzing. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 427–430. IEEE, 2011.

[18] Jerome R Bellegarda. Spoken language understanding for natural interaction: The siri experience. In *Natural Interaction with Robots, Knowbots and Smartphones*, pages 3–14. Springer, 2014.

[19] Oladayo Bello and Sherali Zeadally. Intelligent device-to-device communication in the internet of things. *IEEE Systems Journal*, 10(3):1172–1182, 2016.

[20] Gerald Bieber, Thomas Kirste, and Michael Gaede. Low sampling rate for physical activity recognition. In *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*, page 15. ACM, 2014.

[21] Jiang Bo, Long Xiang, and Gao Xiaopeng. Mobiletest: A tool supporting automatic black box test for software on smart mobile devices. In *Proceedings of the Second International Workshop on Automation of Software Test*, page 8. IEEE Computer Society, 2007.

[22] Szymon Bobek. Context simulator (knowme). `http://glados.kis.agh.edu.pl/`, April 2019.

[23] Szymon Bobek, Sebastian Dziadzio, Paweł Jaciów, Mateusz Ślażyński, and Grzegorz J Nalepa. Understanding context with contextviewer–tool for visualization and initial preprocessing of mobile sensors data. In *International and Interdisciplinary Conference on Modeling and Using Context*, pages 77–90. Springer, 2015.

[24] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[25] Kelly Caine. Local standards for sample size at chi. In *Proceedings of the 2016 CHI conference on human factors in computing systems*, pages 981–992. ACM, 2016.

[26] Huasong Cao, Victor Leung, Cupid Chow, and Henry Chan. Enabling technologies for wireless body area networks: A survey and outlook. *IEEE Communications Magazine*, 47(12):84–93, 2009.

[27] Aaron E Carroll, Linda A DiMeglio, Stephanie Stein, and David G Marrero. Using a cell phone-based glucose monitoring system for adolescent diabetes management. *The Diabetes Educator*, 37(1):59–66, 2011.

[28] Matthew Chalmers. A historical view of context. *Computer Supported Cooperative Work (CSCW)*, 13(3-4):223–247, 2004.

[29] Jiantong Cheng, Ling Yang, Yong Li, and Weihua Zhang. Seamless outdoor/indoor navigation with wifi/gps aided low cost inertial navigation system. *Physical Communication*, 13:31–43, 2014.

[30] David Chu, Nicholas D Lane, Ted Tsung-Te Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li, and Feng Zhao. Balancing energy, latency and accuracy for mobile sensor data classification. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 54–67. ACM, 2011.

[31] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[32] Erik Dahlman, Stefan Parkvall, and Johan Skold. *4G: LTE/LTE-advanced for mobile broadband*. Academic press, 2013.

[33] Marco D'Ambros, Alberto Bacchelli, and Michele Lanza. On the impact of design flaws on software defects. In *2010 10th International Conference on Quality Software*, pages 23–31. IEEE, 2010.

[34] Anind K Dey. Context-aware computing: The cyberdesk project. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, 1998.

[35] Anind K Dey. Context-aware computing. In *Ubiquitous Computing Fundamentals*, pages 335–366. Chapman and Hall/CRC, 2016.

[36] Anind K Dey, Gregory D Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human–Computer Interaction*, 16(2-4):97–166, 2001.

[37] Paul Dourish. What we talk about when we talk about context. *Personal and ubiquitous computing*, 8(1):19–30, 2004.

[38] Joshua J Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A Ridley, and Georg Wicherski. *Android hacker's handbook*. John Wiley & Sons, 2014.

[39] W Keith Edwards, Victoria Bellotti, Anind K Dey, and Mark W Newman. The challenges of user-centered design and evaluation for infrastructure. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 297–304. ACM, 2003.

[40] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Camera-based navigation of a low-cost quadrocopter. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2815–2821. IEEE, 2012.

[41] Rovio Entertainment. Angry birds. `https://www.angrybirds.com/`, April 2019.

[42] Jon Eyolfson, Lin Tan, and Patrick Lam. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162. ACM, 2011.

[43] Inc. Facebook. Facebook. `https://www.facebook.com/`, April 2019.

[44] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopou-

los, Ramesh Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 179–194. ACM, 2010.

[45] Denzil Ferreira, Anind K Dey, and Vassilis Kostakos. Understanding human-smartphone concerns: a study of battery life. In *International Conference on Pervasive Computing*, pages 19–33. Springer, 2011.

[46] Denzil Ferreira, Eija Ferreira, Jorge Goncalves, Vassilis Kostakos, and Anind K Dey. Revisiting human-battery interaction with an interactive battery interface. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 563–572. ACM, 2013.

[47] Denzil Ferreira, Vassilis Kostakos, and Anind K Dey. Aware: mobile context instrumentation framework. *Frontiers in ICT*, 2:6, 2015.

[48] Denzil Ferreira, Christian Schuss, Chu Luo, Jorge Goncalves, Vassilis Kostakos, and Timo Rahkonen. Indoor light scavenging on smartphones. In *Proceedings of the 15th International Conference on Mobile and Ubiquitous Multimedia*, pages 369–371. ACM, 2016.

[49] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Theodoros Anagnostopoulos, Vassilis Kostakos, Chu Luo, and Xiang Su. Sensorclone: a framework for harnessing smart devices with virtual sensors. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 328–338. ACM, 2018.

[50] Huber Flores, Rajesh Sharma, Denzil Ferreira, Chu Luo, Vassilis Kostakos, Sasu Tarkoma, Pan Hui, and Yong Li. Social-aware device-to-device communication: a contribution for edge and fog computing? In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 1466–1471. ACM, 2016.

[51] Huber Flores, Satish Narayana Srirama, and Carlos Paniagua. A generic middleware framework for handling process intensive hybrid cloud services from mobiles. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, pages 87–94. ACM, 2011.

[52] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, 1994.

[53] USB Implementers Forum. Usb. `https://www.usb.org/`, April 2019.

[54] The Linux Foundation. Tizen. `https://www.tizen.org/`, April 2019.

[55] Robin Genuer, Jean-Michel Poggi, and Christine Tuleau-Malot. Variable selection using random forests. *Pattern Recognition Letters*, 31(14):2225–2236, 2010.

[56] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. Reran: Timing-and touch-sensitive record and replay for android. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 72–81. IEEE Press, 2013.

[57] María Gómez, Romain Rouvoy, Bram Adams, and Lionel Seinturier. Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring. In *Mobile Software Engineering and Systems (MOBILESoft), 2016 IEEE/ACM International Conference on*, pages 88–99. IEEE, 2016.

[58] Jorge Goncalves, Zhanna Sarsenbayeva, Niels van Berkel, Chu Luo, Simo Hosio, Sirkka Risanen, Hannu Rintamäki, and Vassilis Kostakos. Tapping task performance on smartphones in cold temperature. *Interacting with Computers*, 29(3):355–367, 2016.

[59] Google. Android studio. `https://developer.android.com/studio`, April 2019.

[60] Google. Art and dalvik. `https://source.android.com/devices/tech/dalvik`, April 2019.

[61] Google. Monkey. `https://developer.android.com/studio/test/monkey`, April 2019.

[62] Google. monkeyrunner. `https://developer.android.com/studio/test/monkeyrunner/`, April 2019.

[63] Google. The official site for android app developers. `https://developer.android.com/`, April 2019.

[64] Google. Sensors overview - android developers. `https://developer.android.com/guide/topics/sensors/sensors_overview/`, April 2019.

[65] Google. Sensors overview, java - android developers. `https://developer.android.com/guide/topics/sensors/sensors_overview#java`, April 2019.

[66] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM*

SIGKDD explorations newsletter, 11(1):10–18, 2009.

[67] Hyung Kil Ham and Young Bom Park. Mobile application compatibility test system design for android fragmentation. In *International Conference on Advanced Software Engineering and Its Applications*, pages 314–320. Springer, 2011.

[68] Dan Han, Chenlei Zhang, Xiaochao Fan, Abram Hindle, Kenny Wong, and Eleni Stroulia. Understanding android fragmentation with topic analysis of vendor-specific bugs. In *2012 19th Working Conference on Reverse Engineering*, pages 83–92. IEEE, 2012.

[69] Shuai Hao, Bin Liu, Suman Nath, William GJ Halfond, and Ramesh Govindan. Puma: programmable ui-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 204–217. ACM, 2014.

[70] Theodore D Hellmann, Abhishek Sharma, Jennifer Ferreira, and Frank Maurer. Agile testing: Past, present, and future–charting a systematic map of testing in agile software development. In *2012 Agile Conference*, pages 55–63. IEEE, 2012.

[71] SY Hui. Planar wireless charging technology for portable electronic products and qi. *Proceedings of the IEEE*, 101(6):1290–1301, 2013.

[72] Antonio Ken Iannillo, Roberto Natella, Domenico Cotroneo, and Cristina Nita-Rotaru. Chizpurfle: A gray-box android fuzzer for vendor service customizations. In *Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on*, pages 1–11. IEEE, 2017.

[73] IEEE. Ieee recommended practice for software requirements specifications. `https://standards.ieee.org/standard/830-1998.html`, April 2019.

[74] UC Irvine. Machine learning repository. `https://archive.ics.uci.edu/ml/index.php`, April 2019.

[75] ISO/IEC/IEEE. Iso/iec/ieee international standard - systems and software engineering – life cycle processes –requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, pages 1–94, Dec 2011.

[76] Nishant Jain, Xiaozhe Fan, Walter Daniel Leon-Salas, and Anne M Lucietto. Extending battery life of smartphones by overcoming idle power consumption using ambient light energy harvesting. In *2018 IEEE International*

Conference on Industrial Technology (ICIT), pages 978–983. IEEE, 2018.

[77] Pankaj Jalote. *A concise introduction to software engineering*. Springer Science & Business Media, 2008.

[78] Won Jeon, Tasneem Brutch, and Simon Gibbs. Webcl for hardware-accelerated web applications. In *TIZEN Developer Conference May*, pages 7–9, 2012.

[79] Wang Jun and Fanpeng Meng. Software testing based on cloud computing. In *2011 International Conference on Internet Computing and Information Services*, pages 176–178. IEEE, 2011.

[80] JUnit. Junit 5. `https://junit.org/junit5/`, April 2019.

[81] Erik Kamsties, Klaus Hörmann, and Maud Schlich. Requirements engineering in small and medium enterprises. *Requirements engineering*, 3(2):84–90, 1998.

[82] Wazir Zada Khan, Yang Xiang, Mohammed Y Aalsalem, and Quratulain Arshad. Mobile phone sensing systems: A survey. *IEEE Communications Surveys & Tutorials*, 15(1):402–427, 2013.

[83] Naoto Kimura and Shahram Latifi. A survey on data compression in wireless sensor networks. In *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, volume 2, pages 8–13. IEEE, 2005.

[84] Jesper Kjeldskov and Connor Graham. A review of mobile hci research methods. In *International Conference on Mobile Human-Computer Interaction*, pages 317–335. Springer, 2003.

[85] Simon Klakegg, Chu Luo, Jorge Goncalves, Simo Hosio, and Vassilis Kostakos. Instrumenting smartphones with portable nirs. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 618–623. ACM, 2016.

[86] Seunghoon Ko, Hyungcheol Shin, Jaemin Lee, Hongjae Jang, Byeong-Cheol So, Ilhyun Yun, and Kwyro Lee. Low noise capacitive sensor for multi-touch mobile handset's applications. In *Solid State Circuits Conference (A-SSCC), 2010 IEEE Asian*, pages 1–4. IEEE, 2010.

[87] Sigmund Ed Koch. Psychology: a study of a science. 1959.

[88] Chatchai Kongaut and Erik Bohlin. Investigating mobile broadband adoption and usage: A case of smartphones in sweden. *Telematics and Informatics*, 33(3):742–752, 2016.

[89] Vassilis Kostakos and Denzil Ferreira. The rise of ubiquitous instrumentation. *Frontiers in ICT*, 2:3, 2015.

[90] Arvind Kumar and Km Pooja. Steganography-a data hiding technique. *International Journal of Computer Applications*, 9(7):19–23, 2010.

[91] Elina Kuosmanen, Valerii Kan, Aku Visuri, Julio Vega, Yuuki Nishiyama, Anind K Dey, Simon Harper, and Denzil Ferreira. Mobile-based monitoring of parkinson's disease. In *Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia*, pages 441–448. ACM, 2018.

[92] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. A survey of mobile phone sensing. *IEEE Communications magazine*, 48(9):140–150, 2010.

[93] Dong Li and Prasun Sinha. Rbtp: Low-power mobile discovery protocol through recursive binary time partitioning. *IEEE Transactions on Mobile Computing*, 13(2):263–273, 2014.

[94] Gang Li, Boon-Leng Lee, and Wan-Young Chung. Smartwatch-based wearable eeg system for driver drowsiness detection. *IEEE Sensors Journal*, 15(12):7169–7180, 2015.

[95] Chieh-Jan Mike Liang, Nicholas D Lane, Niels Brouwers, Li Zhang, Börje F Karlsson, Hao Liu, Yan Liu, Jun Tang, Xiang Shan, Ranveer Chandra, et al. Caiipa: Automated large-scale mobile app testing through contextual fuzzing. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 519–530. ACM, 2014.

[96] Zhifang Liu, Xiaopeng Gao, and Xiang Long. Adaptive random testing of mobile application. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 2, pages V2–297. IEEE, 2010.

[97] Hong Lu, Wei Pan, Nicholas D Lane, Tanzeem Choudhury, and Andrew T Campbell. Soundsense: scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 165–178. ACM, 2009.

[98] Chu Luo, Henri Koski, Mikko Korhonen, Jorge Goncalves, Theodoros

Anagnostopoulos, Shin'Ichi Konomi, Simon Klakegg, and Vassilis Kostakos. Rapid clock synchronisation for ubiquitous sensing services involving multiple smartphones. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pages 476–481. ACM, 2017.

[99] Chu Luo, Miikka Kuutila, Simon Klakegg, Denzil Ferreira, Huber Flores, Jorge Goncalves, Vassilis Kostakos, and Mika Mäntylä. How to validate mobile crowdsourcing design? leveraging data integration in prototype testing. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 1448–1453. ACM, 2016.

[100] Qi Luo, Denys Poshyvanyk, Aswathy Nair, and Mark Grechanik. Forepost: a tool for detecting performance problems with feedback-driven learning software testing. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 593–596. ACM, 2016.

[101] Khushbu Madan and Rajan Yadav. Behavioural intention to adopt mobile wallet: a developing country perspective. *Journal of Indian Business Research*, 8(3):227–244, 2016.

[102] Ralph Maderholz. Computer assisted project management-integrated software development environment delivers project estimation data. In *GI-20. Jahrestagung II*, pages 465–474. Springer, 1990.

[103] Tim A Majchrzak and Matthias Schulte. Context-dependent testing of applications for mobile devices. *Open Journal of Web Technologies (OJWT)*, 2(1):27–39, 2015.

[104] Mirza Aamir Mehmood, MNA Khan, and Wasif Afzal. Automating test data generation for testing context-aware applications. In *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pages 104–108. IEEE, 2018.

[105] Zhanshuai Meng, Yanyan Jiang, and Chang Xu. Facilitating reusable and scalable automated testing and analysis for android apps. In *Proceedings of the 7th Asia-Pacific Symposium on Internetware*, pages 166–175. ACM, 2015.

[106] Desta Mengistu, Hangjung Zo, and Jae Jeung Rho. M-government: opportunities and challenges to deliver mobile government services in developing countries. In *2009 Fourth International Conference on Computer Sciences and*

Convergence Information Technology, pages 1445–1450. IEEE, 2009.

[107] Microsoft. Windows 10 and windows 10 mobile. `https://docs.microsoft.com/en-us/windows/windows-10/`, April 2019.

[108] Charlie Miller, Dion Blazakis, Dino DaiZovi, Stefan Esser, Vincenzo Iozzo, and Ralf-Philip Weinmann. *iOS Hacker's Handbook*. John Wiley & Sons, 2012.

[109] Chulhong Min, Seungwoo Kang, Chungkuk Yoo, Jeehoon Cha, Sangwon Choi, Younghan Oh, and Junehwa Song. Exploring current practices for battery use and management of smartwatches. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 11–18. ACM, 2015.

[110] Chulhong Min, Seungchul Lee, Changhun Lee, Youngki Lee, Seungwoo Kang, Seungpyo Choi, Wonjung Kim, and Junehwa Song. Pada: power-aware development assistant for mobile sensing applications. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 946–957. ACM, 2016.

[111] Gary Mogyorodi. Requirements-based testing: an overview. In *Proceedings 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems. TOOLS 39*, pages 286–295. IEEE, 2001.

[112] Robert K Morrow. *Wireless network coexistence*. McGraw-Hill New York, NY, USA:, 2004.

[113] Madan Musuvathi and Shaz Qadeer. Chess: Systematic stress testing of concurrent software. In *International Symposium on Logic-based Program Synthesis and Transformation*, pages 15–16. Springer, 2006.

[114] Feng Nan, Joseph Wang, and Venkatesh Saligrama. Pruning random forests for prediction on a budget. In *Advances in neural information processing systems*, pages 2334–2342, 2016.

[115] Chris Otto, Aleksandar Milenkovic, Corey Sanders, and Emil Jovanov. System architecture of a wireless body area sensor network for ubiquitous health monitoring. *Journal of mobile multimedia*, 1(4):307–326, 2006.

[116] Je-Ho Park, Young Bom Park, and Hyung Kil Ham. Fragmentation problem in android. In *2013 International Conference on Information Science and Applications (ICISA)*, pages 1–2. IEEE, 2013.

[117] Mattias Patzold. Countdown for the full-scale development of 5g new radio [mobile radio]. *IEEE Vehicular Technology Magazine*, 13(2):7–13, 2018.

[118] William E Perry. *Effective methods for software testing: Includes complete guidelines, Checklists, and Templates*. John Wiley & Sons, 2007.

[119] S Phad Vitthal, S Bhosale Rajkumar, and R Panhalkar Archana. A novel security scheme for secret data using cryptography and steganography. *IJ Computer Network and Information Security*, 2:36–42, 2012.

[120] Jurairat Phuttharak and Seng W Loke. A review of mobile crowdsourcing architectures and challenges: Toward crowd-empowered internet-of-things. *IEEE Access*, 7:304–324, 2019.

[121] Martin Pielot. Large-scale evaluation of call-availability prediction. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 933–937. ACM, 2014.

[122] Martin Pielot, Rodrigo De Oliveira, Haewoon Kwak, and Nuria Oliver. Didn't you see my message?: predicting attentiveness to mobile instant messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3319–3328. ACM, 2014.

[123] Benjamin Poppinga, Wilko Heuten, and Susanne Boll. Sensor-based identification of opportune moments for triggering notifications. *IEEE Pervasive Computing*, 13(1):22–29, 2014.

[124] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 347–356. ACM, 2010.

[125] Davy Preuveneers and Yolande Berbers. Towards context-aware and resource-driven self-adaptation for mobile handheld applications. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 1165–1170. ACM, 2007.

[126] Zhengrui Qin, Yutao Tang, Ed Novak, and Qun Li. Mobiplay: A remote execution based record-and-replay tool for mobile applications. In *Proceedings of the 38th International Conference on Software Engineering*, pages 571–582. ACM, 2016.

[127] Qualcomm. Snapdragon 845 mobile platform. `https://www.qualcomm.`

com/products/snapdragon-845-mobile-platform/, April 2019.

[128] Kiran K Rachuri, Christos Efstratiou, Ilias Leontiadis, Cecilia Mascolo, and Peter J Rentfrow. Metis: Exploring mobile phone sensing offloading for efficiently supporting social sensing applications. In *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 85–93. IEEE, 2013.

[129] Omer Rashid, Paul Coulton, and Reuben Edwards. Providing location based information/advertising for existing mobile phone users. *Personal and Ubiquitous Computing*, 12(1):3–10, 2008.

[130] Rapeepat Ratasuk, Benny Vejlgaard, Nitin Mangalvedhe, and Amitava Ghosh. Nb-iot system for m2m communication. In *2016 IEEE wireless communications and networking conference*, pages 1–5. IEEE, 2016.

[131] Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 190–203. ACM, 2014.

[132] Bernd Resch, Anja Summa, Günther Sagl, Peter Zeile, and Jan-Philipp Exner. Urban emotions-geo-semantic emotion extraction from technical sensors, human sensors and crowdsourced data. In *Progress in location-based services 2014*, pages 199–212. Springer, 2015.

[133] Research and Markets. Global smartwatch market size, market share, application analysis, regional outlook, growth trends, key players, competitive strategies and forecasts, 2018 to 2026. https://www.researchandmarkets.com/research/7vvn28/global_smartwatch?w=12, April 2019.

[134] Nick Ryan, Jason Pascoe, and David Morse. Enhanced reality fieldwork: the context aware archaeological assistant. *Bar International Series*, 750:269–274, 1999.

[135] Dipti Kapoor Sarmah and Neha Bajpai. Proposed system for data hiding using cryptography and steganography. *International Journal of Computer Applications*, 8(9):7–10, 2010.

[136] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and develop-*

ment in information retrieval, pages 253–260. ACM, 2002.

[137] B Schilit, N Adams, and R Want. Context-aware computing applications. In *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

[138] Bill N Schilit and Marvin M Theimer. Disseminating active map information to mobile hosts. *IEEE network*, 8(5):22–32, 1994.

[139] Sakura She, Sasindran Sivapalan, and Ian Warren. Hermes: A tool for testing mobile device applications. In *Software Engineering Conference, 2009. ASWEC'09. Australian*, pages 121–130. IEEE, 2009.

[140] Muhammad Shoaib, Hans Scholten, Paul JM Havinga, and Ozlem Durmaz Incel. A hierarchical lazy smoking detection algorithm using smartwatch sensors. In *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pages 1–6. IEEE, 2016.

[141] Bluetooth SIG. Bluetooth. `https://www.bluetooth.com/`, April 2019.

[142] Yogesh Singh. *Software testing*. Cambridge University Press Cambridge, 2012.

[143] Andrew L Skinner, Christopher J Stone, Hazel Doughty, and Marcus R Munafò. Stopwatch: The preliminary evaluation of a smartwatch-based system for passive detection of cigarette smoking. *Nicotine and Tobacco Research*, 21(2):257–261, 2018.

[144] Brian A Smith, Xiaojun Bi, and Shumin Zhai. Optimizing touchscreen keyboards for gesture typing. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3365–3374. ACM, 2015.

[145] Kwangsik Song, Ah-Rim Han, Sehun Jeong, and Sung Deok Cha. Generating various contexts from permissions for testing android applications. In *SEKE*, pages 87–92, 2015.

[146] Sean Stolberg. Enabling agile testing through continuous integration. In *2009 Agile Conference*, pages 369–374. IEEE, 2009.

[147] Boyuan Sun, Qiang Ma, Shanfeng Zhang, Kebin Liu, and Yunhao Liu. iself: Towards cold-start emotion labeling using transfer learning with smartphones. *ACM Transactions on Sensor Networks (TOSN)*, 13(4):30, 2017.

[148] San-Tsai Sun, Andrea Cuadros, and Konstantin Beznosov. Android rooting:

Methods, detection, and evasion. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 3–14. ACM, 2015.

[149] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.

[150] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[151] Amir E Sarabadani Tafreshi and Moira C Norrie. Screenpress: a powerful and flexible platform for networked pervasive display systems. In *Proceedings of the 6th ACM International Symposium on Pervasive Displays*, page 13. ACM, 2017.

[152] Ralf Tonjes, Eike Steffen Reetz, Marten Fischer, and Daniel Kuemper. Automated testing of context-aware applications. In *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*, pages 1–5. IEEE, 2015.

[153] Asmau Usman, Noraini Ibrahim, and Ibrahim Anka Salihu. Test case generation from android mobile applications focusing on context events. In *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, pages 25–30. ACM, 2018.

[154] W3C. Extensible markup language (xml). `https://www.w3.org/XML/`, April 2019.

[155] Daniel Wagner and Dieter Schmalstieg. Making augmented reality practical on mobile phones, part 1. *IEEE Computer Graphics and Applications*, 29(3), 2009.

[156] Lili Wei, Yepang Liu, and Shing-Chi Cheung. Taming android fragmentation: Characterizing and detecting compatibility issues for android apps. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 226–237. IEEE, 2016.

[157] Hongyi Wen, Julian Ramos Rojas, and Anind K Dey. Serendipity: Finger gesture recognition using an off-the-shelf smartwatch. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3847–3851. ACM, 2016.

[158] Edgardo Barsallo Yi, Amiya Maji, and Saurabh Bagchi. How reliable is my wearable: A fuzz testing-based study. In *2018 48th Annual IEEE/IFIP*

International Conference on Dependable Systems and Networks (DSN), pages 410–417. IEEE, 2018.

[159] Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[160] Chuang-Wen You, Nicholas D Lane, Fanglin Chen, Rui Wang, Zhenyu Chen, Thomas J Bao, Martha Montes-de Oca, Yuting Cheng, Mu Lin, Lorenzo Torresani, et al. Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 13–26. ACM, 2013.

[161] Siena Yu and Shingo Takada. External event-based test cases for mobile application. In *8th International C Conference on Computer Science and Software Engineering, C3S2E 2015*, pages 148–149. Association for Computing Machinery, 2015.

[162] BB Zaidan, AA Zaidan, AK Al-Frajat, and HA Jalab. On the differences between hiding information and cryptography techniques: An overview. *Journal of Applied Sciences(Faisalabad)*, 10(15):1650–1655, 2010.

[163] Cheng Zhang, Juyuan Yang, Yi Zhang, Jing Fan, Xin Zhang, Jianjun Zhao, and Peizhao Ou. Automatic parameter recommendation for practical api usage. In *Proceedings of the 34th International Conference on Software Engineering*, pages 826–836. IEEE Press, 2012.

[164] Haochen Zhang, Minyen Kan, Yiqun Liu, and Shaoping Ma. Online social network profile linkage based on cost-sensitive feature acquisition. In *Chinese National Conference on Social Media Processing*, pages 117–128. Springer, 2014.

[165] Li Lyna Zhang, Chieh-Jan Mike Liang, Wei Zhang, and Enhong Chen. Towards a contextual and scalable automated-testing service for mobile apps. In *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*, pages 97–102. ACM, 2017.

[166] Yu Zhang, Tao Gu, Chu Luo, Vassilis Kostakos, and Aruna Seneviratne. Findroidhr: Smartwatch gesture input with optical heartrate monitor. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1):56, 2018.